Gebze Technical University
ComputerEngineering

CSE 222 -2018 Spring

HOMEWORK 4 REPORT

YUNUS ÇEVİK
141044080

Course Assistant: MEHMET BURAK KOCA

1 INTRODUCTION

1.1 Problem Definition

1.1.1 Part1

Binary Tree sınıfından extend ettiğimiz generic bir sınıf oluşturup. Bu sınıf içerisinde bizden istenen add, levelOrderSearch, postOrderSearch metotlarının çalışma şekillerini Binary Tree gibi değil, General Tree gibi implement etmemiz istenmektedir. Ayrıca Binary Tree sınıfında içerisinde yer alan preOrderTraverse metodunu da Override etmemiz istenmiştir.

Not: Parent Node' nun sol node u Child Node olmalıdır. Ayrıca Parent' ın bir Child Node' u varsa yeni eklenecek Child Node önceki Child Node'un sağ node una eklemiz istenmektedir.

1.1.2 Part2

Multi Dimension Tree yapısını oluşturmak için ders kitabının kaynak kodlarından BinaryTree ile extend, SearchTree ile implement ettiğimiz generic bir sınıf oluşturup. Bu sınıf içerisinde bizden SearchTree interface yapısından gelen metotları BinaryTree sınıfını kullanarak Multi Dimension yapıda implement etmemiz istenmektedir. Ayrıca BinaryTree içerisinde bulunan Node yapısı içerisinde tek bir eleman barındırmaktadır. Ancak bizim Node yapımızda birden fazla değeri barındıran Multi Dİmension tasarımı yapılmalıdır.

1.2 System Requirements

Proje, Intellij projesi olarak oluşturulmalıdır. Java 8 uyumlu olmalıdır. Windows ve Linux ortamlarında yüklü Intellij IDE' sinde çalışabilir olmalıdır. Ayrıca projeyi test etmek için bizlere sunulan Virtual Machine içinde bulunan Intellij IDE' sinde de test edilebilir olmalıdır. Benim oluşturmuş olduğum proje bu şartlara uymaktadır. Projede belirtilen part - part ayrılmış kısımların gereksinimleri aşağıda anlatılmaktadır.

1.2.1 Part1

Part1 için gerekli olanlar:

- Proje dosyamızın içinde Part1.java class' ı oluşturarak ders kitabının kaynak kodlarından BinaryTree.java class' ı extend edilmelidir.

- Olusturduğumuz Part1 class' ı içerisinde Binary Tree yapıda olup General Tree gibi çalışan ve bizden istenen metotlar bulunmalıdır.

  Bu metotların prototipleri şöyle olmalıdır.

  - ```
    public boolean add(E parentItem, E childItem,int levelOrPost);
    ```
  - ```
    private Node<E> levelOrderSearch(Node<E> node, E parentItem, boolean print);
    ```
  - ```
    private Node<E> postOrderSearch(Node<E> node, E parentItem)
    ```
  - ```
    @Override
    protected void preOrderTraverse(Node<E> node, int depth, StringBuilder sb);
    ```

1.2.2 Part2

- Proje dosyamızın içinde Part2.java class' ı oluşturarak ders kitabının kaynak kodlarından BinaryTree.java class' ı extend, SearchTree.java interface'ini implement edilmelidir.
- Olusturduğumuz Part2 class' ı içerisinde Binary Tree yapısını kullanarak SearchTree interface yapısında bulunan metotların implementasyonu bizden istenmektedir.

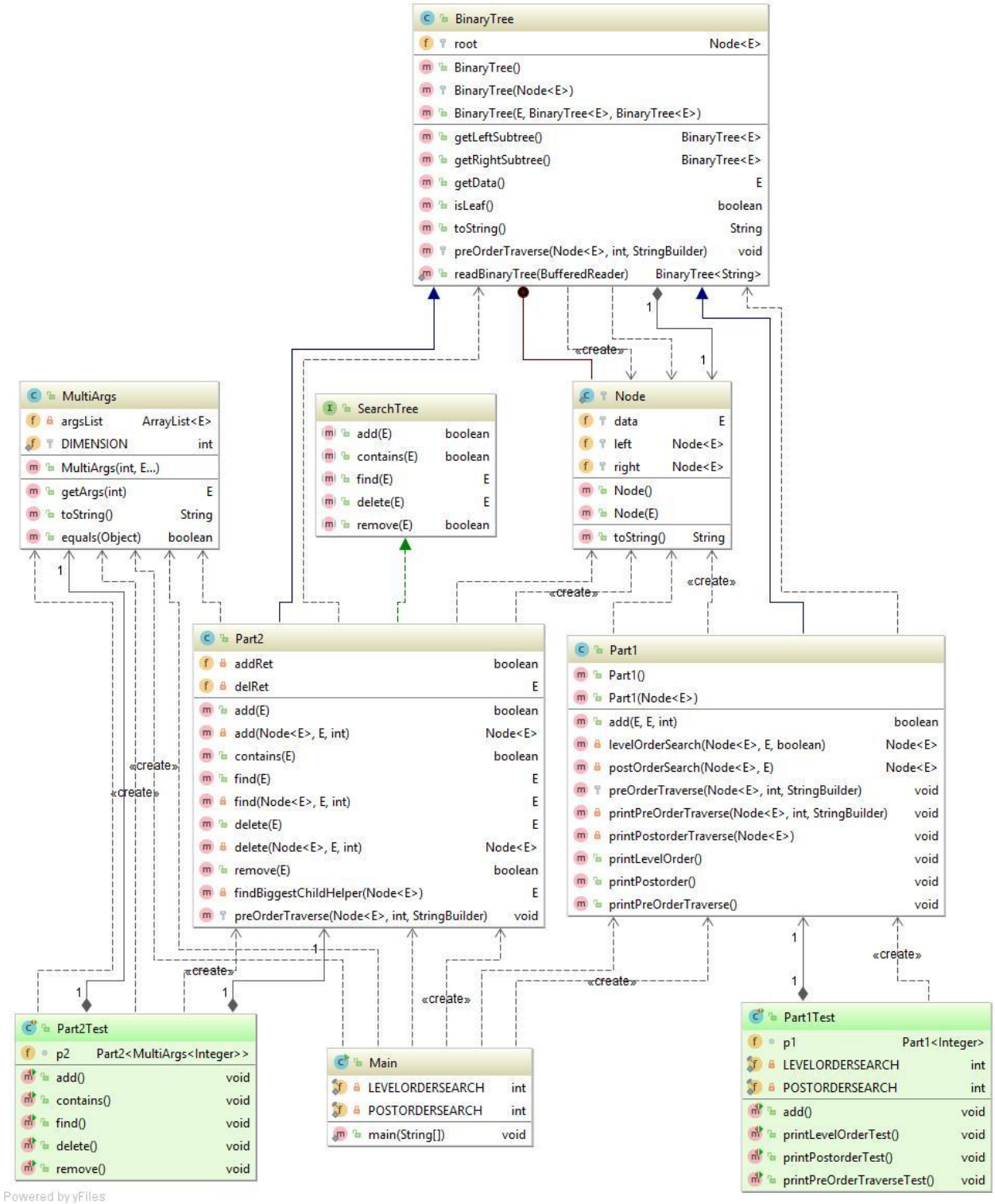
  Bu metotların prototipleri şöyle olmalıdır.

  - ```
    boolean add(E item);
    ```
  - ```
    boolean contains(E target);
    ```
  - ```
    E find(E target);
    ```
  - ```
    E delete(E target);
    ```
  - ```
    boolean remove(E target);
    ```

Not: Yukarıda belirtilen Part1 ve Part2 sınıflarının gerekli metodları ilerleyen bölümlerde detaylı anlatılacaktır.
    ```

2 METHOD

2.1 Class Diagrams



2.1.1 Class Diagram Documantation:

2.1.1.1 Part1

public class Part1<E> extends BinaryTree<E> => Binary Tree yapısını General Tree olarak kullanan ve ayrıca Binary Tree metotları ile sonradan implement edilen add, levelOrderSearch, postOrderSearch ve preOrderTraverse metotlarını kullanma imkanı sağlayan bir sınıftır.

public boolean add(E parentItem, E childItem,int levelOrPost) => Bir ağaç yapısına Node eklemek için kullanılır.

private Node<E> levelOrderSearch(Node<E> node, E parentItem, boolean print) => Binary Tree yapisini General Tree olarak level level olarak search etmeye yarayan metottur.

private Node<E> postOrderSearch(Node<E> node, E parentItem) => Binary Tree yapisini General Tree olarak Post Order Traverse olarak search etmeye yarayan metottur.

@Override
protected void preOrderTraverse(Node<E> node, int depth, StringBuilder sb) => Perform a preorder traversal. (BinaryTree sınıfından alınma.)

private void printPreOrderTraverse(Node<E> node, int depth, StringBuilder sb) => Pre Order Traverse olarak gezinip ekrana hangi nodelardan geçtiğini gösterir

private void printPostorderTraverse(Node<E> node) => Post Order Traverse olarak gezinerek hangi node degerleri uzerinden geçtiğini ekrana basar.

public void printLevelOrder() => Level Order Traverse olarak ekrana cikti verir.

public void printPostorder() => Post Order Traverse olarak ekrana cikti verir.

public void printPreOrderTraverse() => Pre Order Traverse olarak ekrana cikti verir.

2.1.1.2 Part2

public class Part2< E extends MultiArgs> extends BinaryTree<E> implements SearchTree<E> => Binary Tree yapısını Multi Dimension Tree olarak oluşturup Search Tree interface içindeki metotların implementinin sağlandığı bir sınıftır.

public boolean add(E item) => Multi Dimension ağaç yapısına bir eleman eklemek için kullanılan metottur.

private Node<E> add(Node<E> node, E item, int planeLine) => Recursive çalışır ve olarak aldığı elemanı ağaçta bir Node' a ekler. Ancak Binary Tree üzerinde eklenecek node sag tarafa mı sol tarafa mı olduğunu planeLine parametresi belirler. Her plane de Node un gösterdiği liste üzerinden item değerinin listesindeki değerler karsılaştırılır. Böyle-ce ne tarafa ekleyeceği belirlenir.

public boolean contains(E target) => Ağaç üzerinde belirtilen elemanın olup olmadığını kontrol eder.

public E find(E target) => Ağaç üzerinde search işlemi yapar ve aranan elemanı bulup node' unun listesini döndürür.

private E find(Node<E> node, E target, int planeLine) => Recursive olarak çalışır ve istenen elemanı ağaç üzerinde arar ve node' un gösterdiği listeyi verir.

public E delete(E target)=> Ağaç üzerinde silme işlemi yapar ve sildiği elemenin node' unun listesini dondurur.

private Node<E> delete(Node<E> node, E item , int planeLine) => Recursive olarak çalışır. Elemanı ağaç üzerinde arar ve bulduğu zaman siler. Sildiği elemanın Node' unu dondurur. Ayrıca silinen elemanın node' una bağlı diğer node'lar, bir üstte bulunan node yapısına planeLine ile değerine göre karşılaştırma yapar ve diğer node'larin sağ tarafta mı sol tarafta mı olacağı tekrardan belirlenir.

public boolean remove(E target)=> Ağaç üzerinde silme işlemi yapar.

private E findBiggestChildHelper(Node<E> node) => Silme işleminde en büyük çocuk değerini bulmak için kullanılan recursive çalışan helper metot.

2.1.1.2.1 MultiArgs

public class MultiArgs<E extends Comparable> => Helper Class. Bir cok veriyi bir node içinde tutan sınıf.

public MultiArgs(int dimensionValue, E... args) throws Exception => Constructor'in aldığı parametrelere göre bir node içinde arrayList tutarak birden fazla değer barındırılmış-tır.

@Override
public boolean equals(Object o) => Node'larin gösterdiği listelerin karşılaştırılması.

2.2 Use Case Diagrams

Add use case diagrams if required.(Gerekmemektedir.)

2.3 OtherDiagrams (optional)

Add other diagrams if required. (Gerekmemektedir.)

2.4 Problem Solution Approach

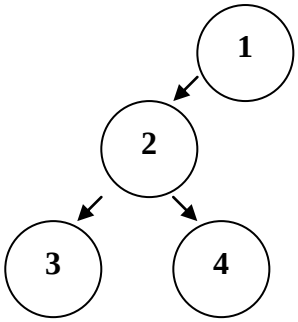
2.4.1 Part1

- Part1 sınıfına BinaryTree sınıfından extend yapıldığında, BinaryTree sınıfı içerisinde bulunan Node yapısını kullanarak bizden istenen metotlar implement edilmektedir.
- İmplement etmeden önce BinaryTree sınıfı içersinede bulunan Node sınıfına parametre almayan Constructor ekledim. Ayrıca bizden override etmemizi istedikleri preOrderTraverse metodu kitabın kaynak kodu olan BinaryTree sınıfında private olarak verilmiş. Ancak private metod override edilemez. Bu nedenle bende BinaryTree içerisindeki private ifadesini protected ifadesine çevirdim ve override işlemini gerçekleştirdim.
- İmplement ettiğimiz Part1 sınıfı üzerinde ekleme ve gezinerek arama yapmamız için şu

metotlar kullanılarak yapılır.

Bu metotlar:

- **boolean add(E parentItem, E childItem, int levelOrPost) =>** Aldığı parametre değerleri ile ağaç yapısına ekleme işlemi yapar. “parentItem” parametresi hangi node üzerinde işlem yapılacağını belirtir. “childItem” parametresi ise belirlenen node üzerinde hangi kola bu değerin ekleneceğini belirler. Belirlenen node üzerine hiçbir node bağlı değilse sol tarafa childItem değerinde node bağlanır. Belirlenen node üzerinin sol tarafında node varsa ve yeni childItem eklenecekse sol taraftaki node un sağ tarafına yeni bir node bağlanır ve childItem eklenir. “levelOrPost” parametresi ise hangi search metodu ile ağaç üzerinde search işlemi yapacağını gösterir. Eğer bu parametreye 1 değeri gelir ise Level Order Search yapar. 2 değeri gelir ise Post Order Search işlemi ile search işlemini gerçekleştirir.



Parent = 1	Child = 2 ve 4
Parent = 2	Child = 3

- **Node<E> levelOrderSearch(Node<E> node, E parentItem, boolean print) =>** Aldığı parametreler ile Binary Tree üzerinde General Tree gibi search işlemi yapmaya yarayan metottur. “node” parametresi ile aldığı bir node un sağ ve sol değerlerine bakarak level order traverse ile gezinir. “parentItem” parametresi ise node değeri üzerinde gezinirken istenen parentItem değerine denk bir node u return etmesine yarar. “print” parametresi ile ekrana çıktı verip verilmeyeceği belirlenir. Bu parametre true ise search işlemi yapmadan ekrana çıktı verir. Ancak parametre false ise parentItem değerine göre search işlemi gerçekleştirir.
- **postOrderSearch(Node<E> node, E parentItem) =>** Aldığı parametreler ile Binary Tree üzerinde General Tree gibi search işlemi yapmaya yarayan metottur. “node” parametresi ile aldığı bir node un sol ve sağ değerlerine bakarak post order traverse ile gezinir. “parentItem” parametresi ise node değeri üzerinde gezinirken istenen parentItem değerine denk bir node u return etmesine yarar.

- **void preOrderTraverse(Node<E> node, int depth, StringBuilder sb)=>** Override edilmiş olan bu metod ile pre order traverse işlemi ile tarama yapar ve her üzerinden geçtiği node un değerini “sb” parametresi ile Call By Referance ile dışa dönderir. “dept” parametresi ile ekrana aktarılacak değerlerin derinlikleri ayarlanır ve ağaç yapısına benzer şekilde çıktı sağlanır.

2.4.2 Part2

- Multi Dimension Tree yapısını oluşturmak için ders kitabının kaynak kodlarından BinaryTree ile extend, SearchTree ile implement ettiğimiz generic bir sınıf oluşturup. Bu sınıf içerisinde bizden SearchTree interface yapısından gelen metotları BinaryTree sınıfını kullanarak Multi Dimension yapıda implement etmek için yardımcı bir sınıf yazılır.
- Yardımcı sınıf MultiArgs.java sınıfıdır. Bu sınıfı oluştururken **public interface SearchTree < E extends Comparable>** içerisinde bulunan **“extends Comparable”** ifadesini silip yardımcı sınıf içerisine yazdım. **public class MultiArgs<E extends Comparable>.**
- Multi Dİmension Tree yapısını temsil edecek Part2.java sınıfını da şu şekilde extend ve implement ettim. **“public class Part2< E extends MultiArgs> extends BinaryTree<E> implements SearchTree<E>”.**
- MultiArgs yardımcı sınıfı içerisinde bir dimension degeri alan ve multi arguman alan parametrelerle birlikte alınan değerler ArrayList içerisine aktarılıp BinaryTree içerisinde bulunan her Node’ un temsil ettiği bir birden fazla değere sahip bir dimension listesi oluşturulmuş oldu.

3 RESULT

3.1 Test Cases

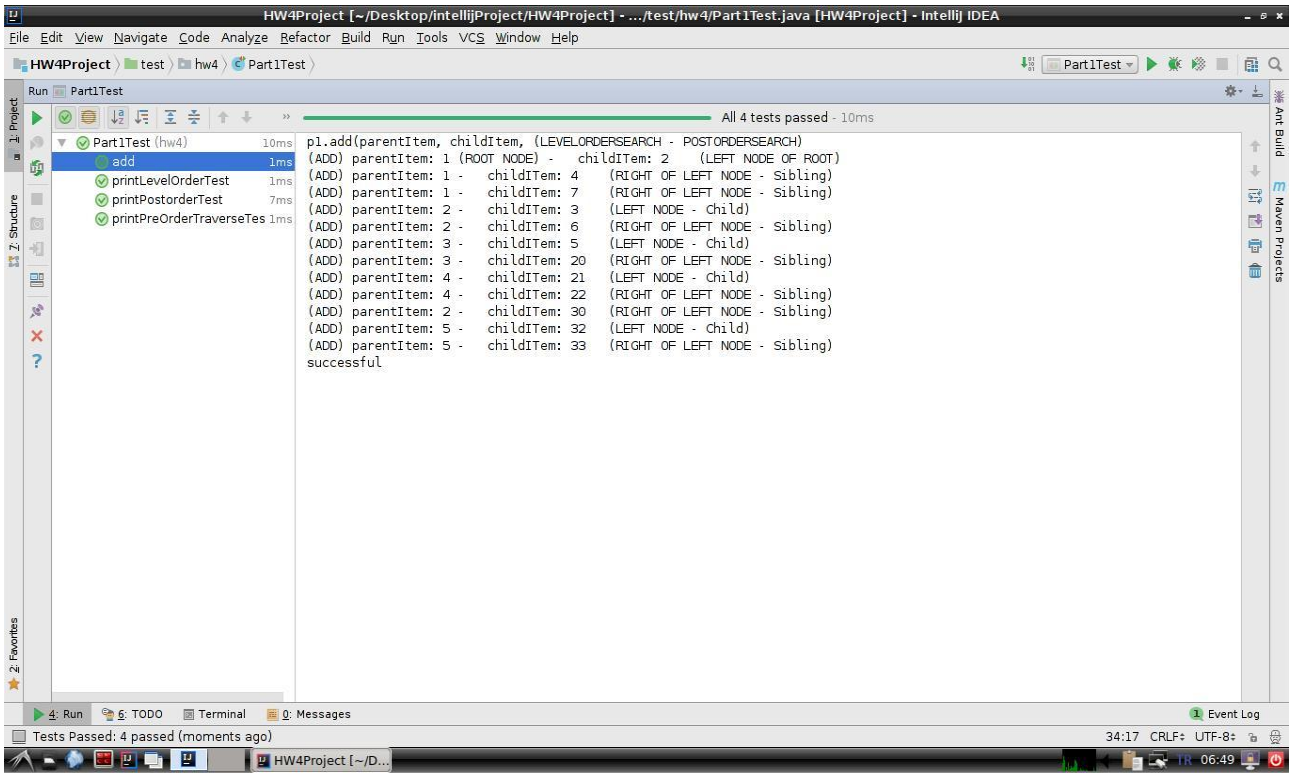
3.1.1 Main Tets

Main Test için projede bulunan main metodu yazılmıştır. Tüm partlarda bizden istenen metotlar tek tek kullanılarak gösterilmiştir. Screen Shot olarak Running Results bölümünde gösterilecektir.

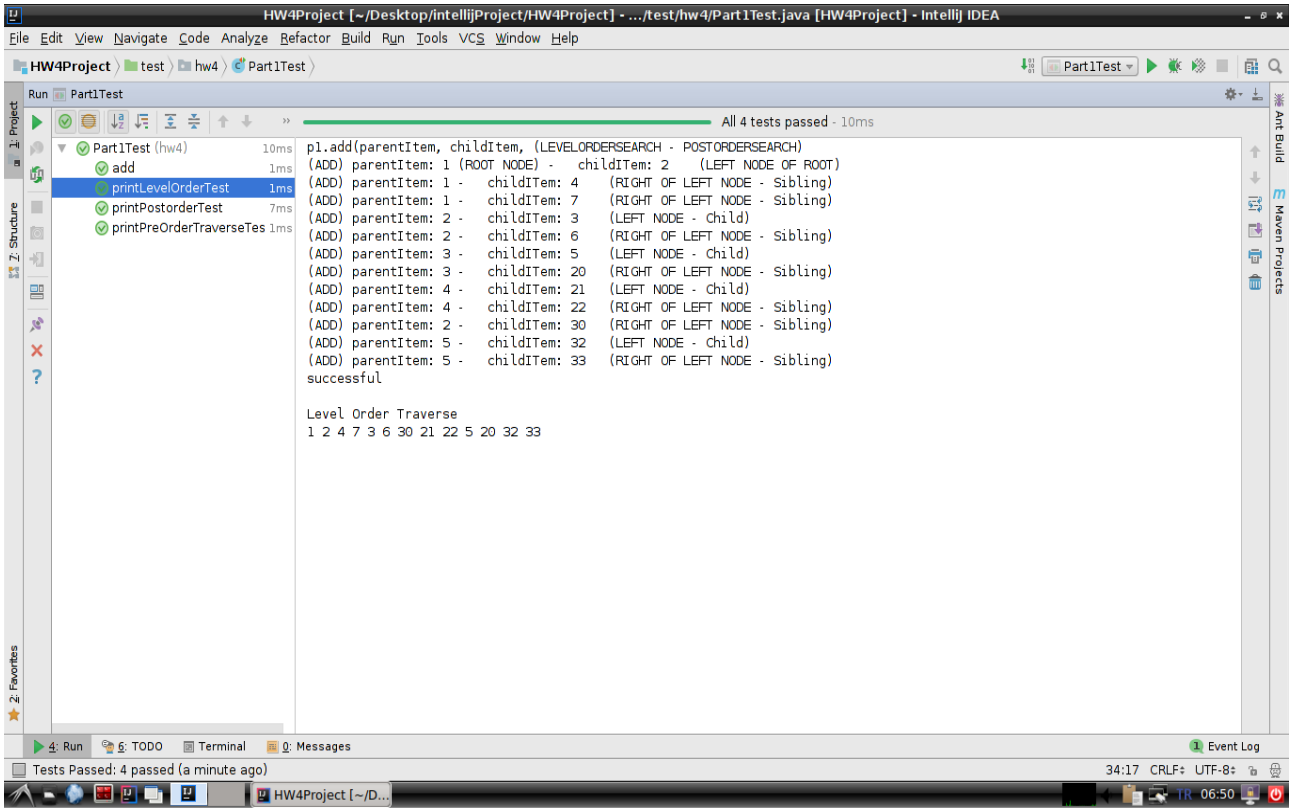
3.1.2 Unit Tests

3.1.2.1 Part1

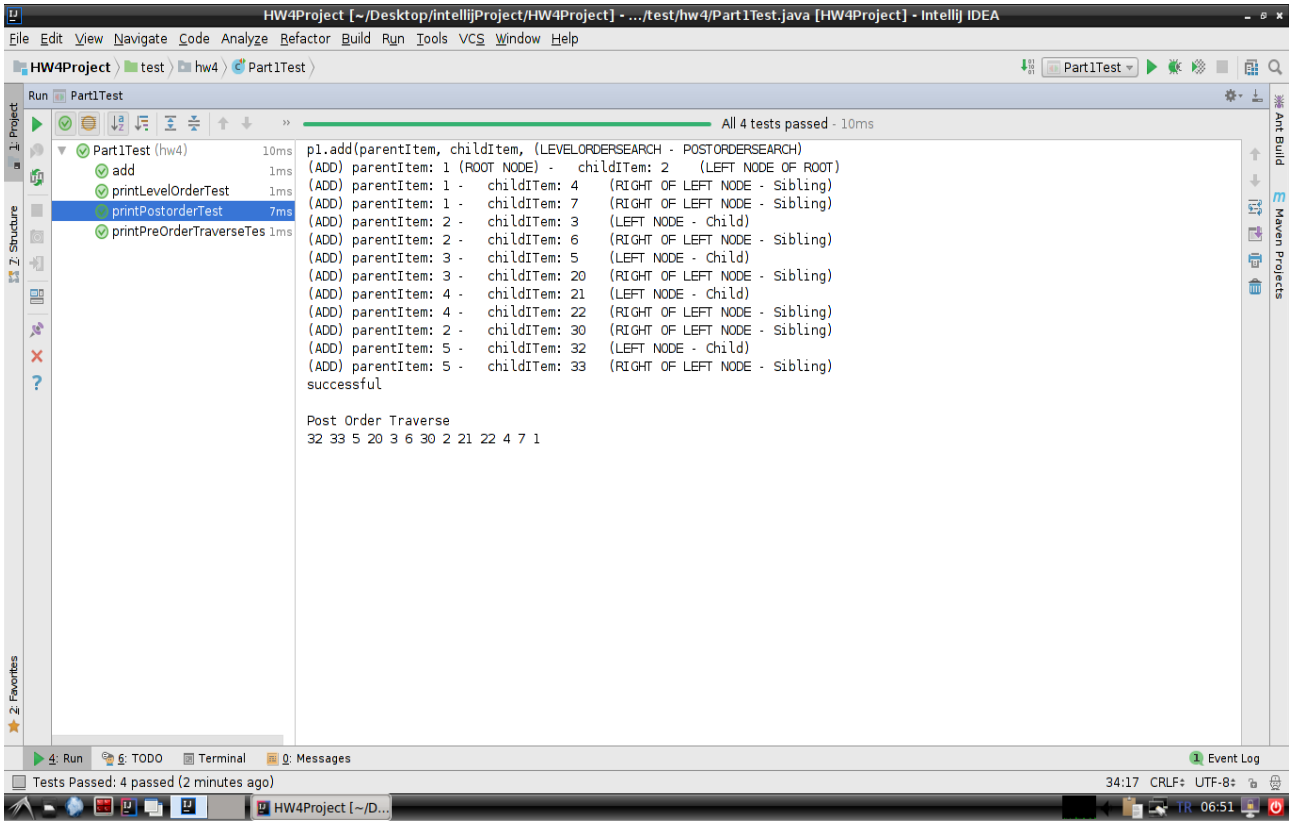
p1.add(parentItem, childItem, (LEVELORDERSEARCH - POSTORDERSEARCH)



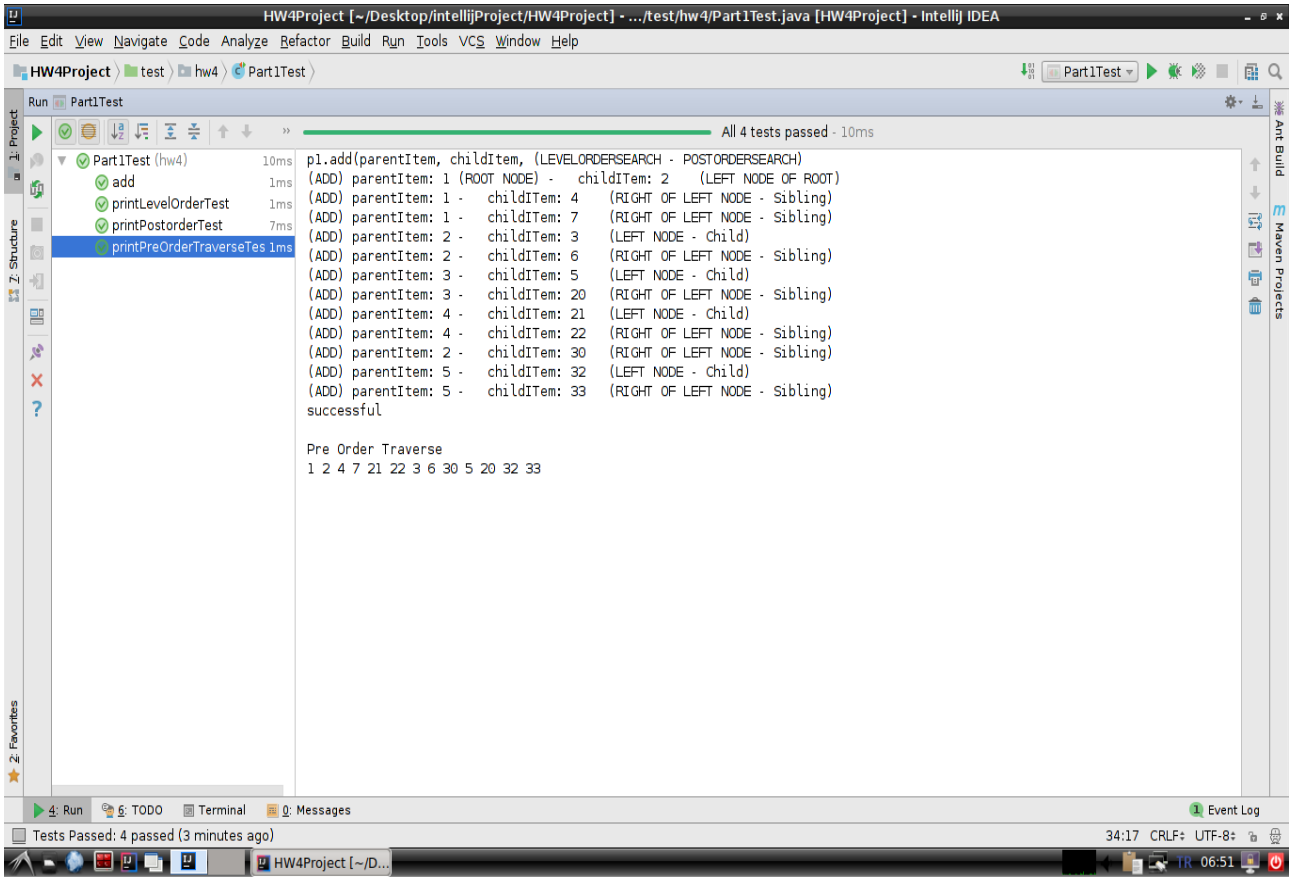
p1.printLevelOrder()



p1.printPostorder()

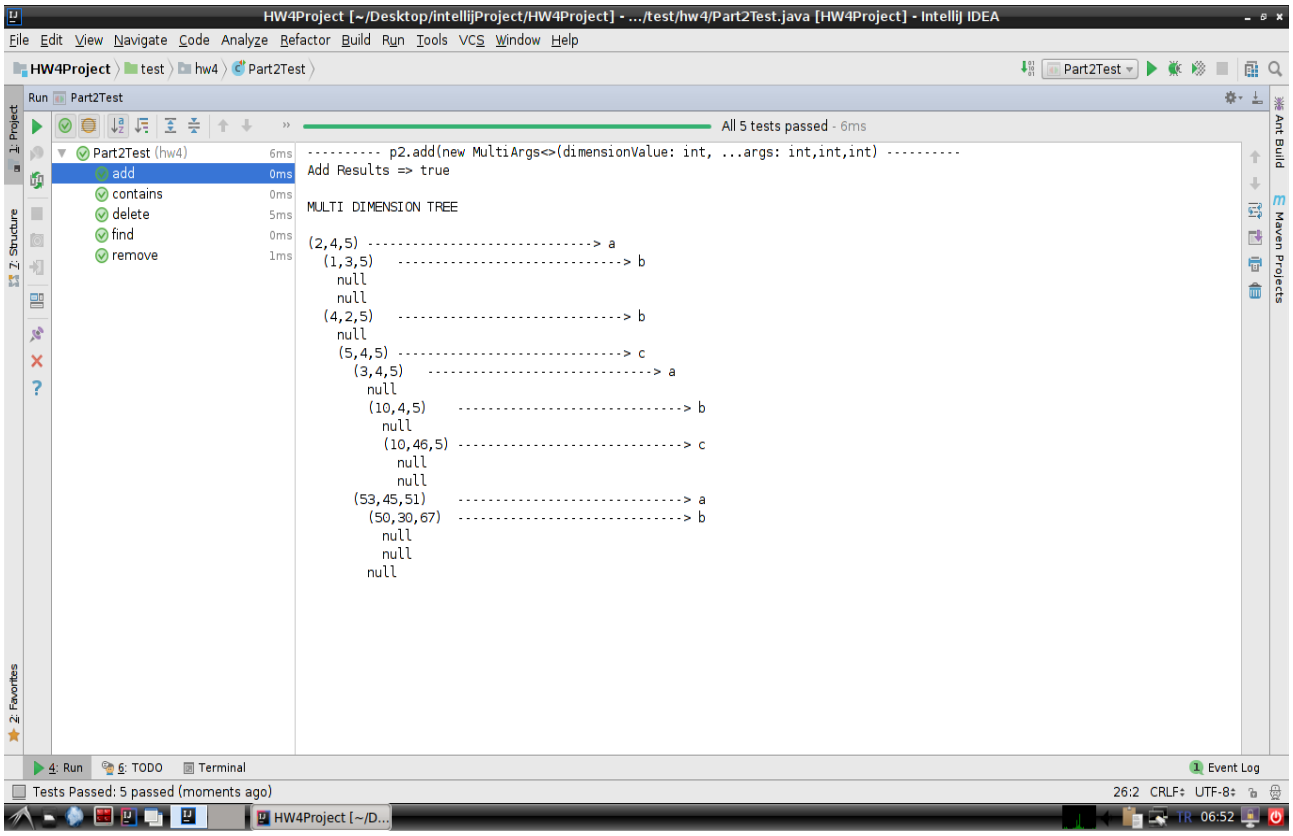


p1.printPreOrderTraverse()

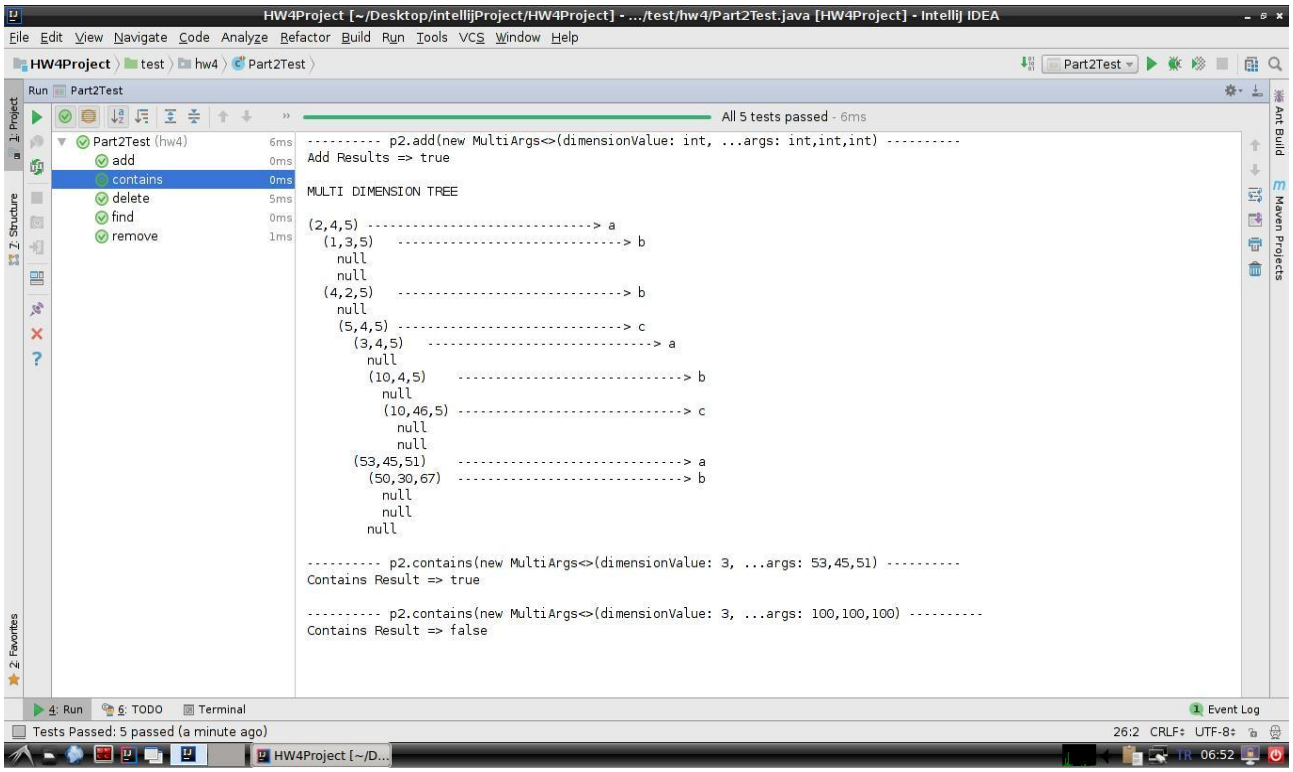


3.1.2.2 Part2

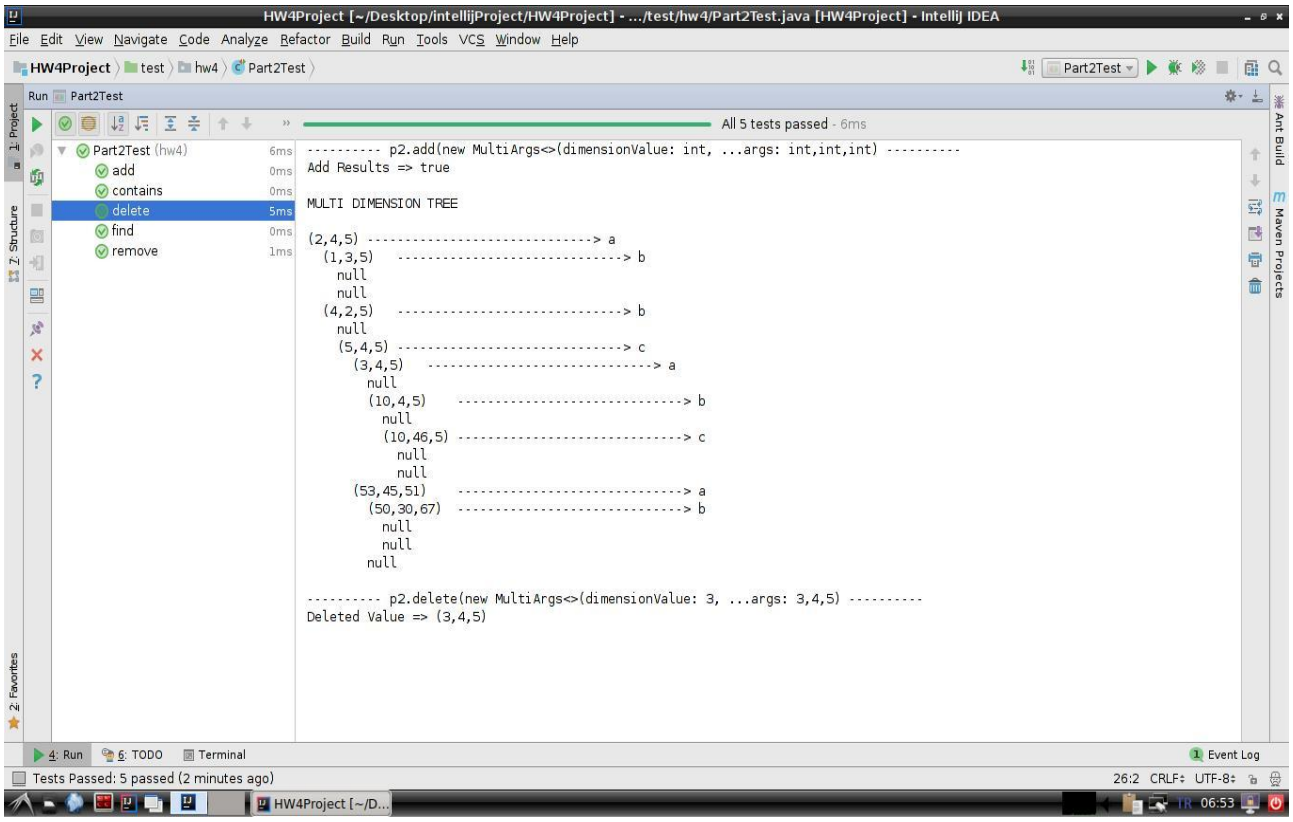
p2.add(new MultiArgs<>(dimensionValue: int, ...args: int,int,int)



p2.contains(new MultiArgs<>(dimensionValue: 3, ...args: 53,45,51)
p2.contains(new MultiArgs<>(dimensionValue: 3, ...args: 100,100,100)

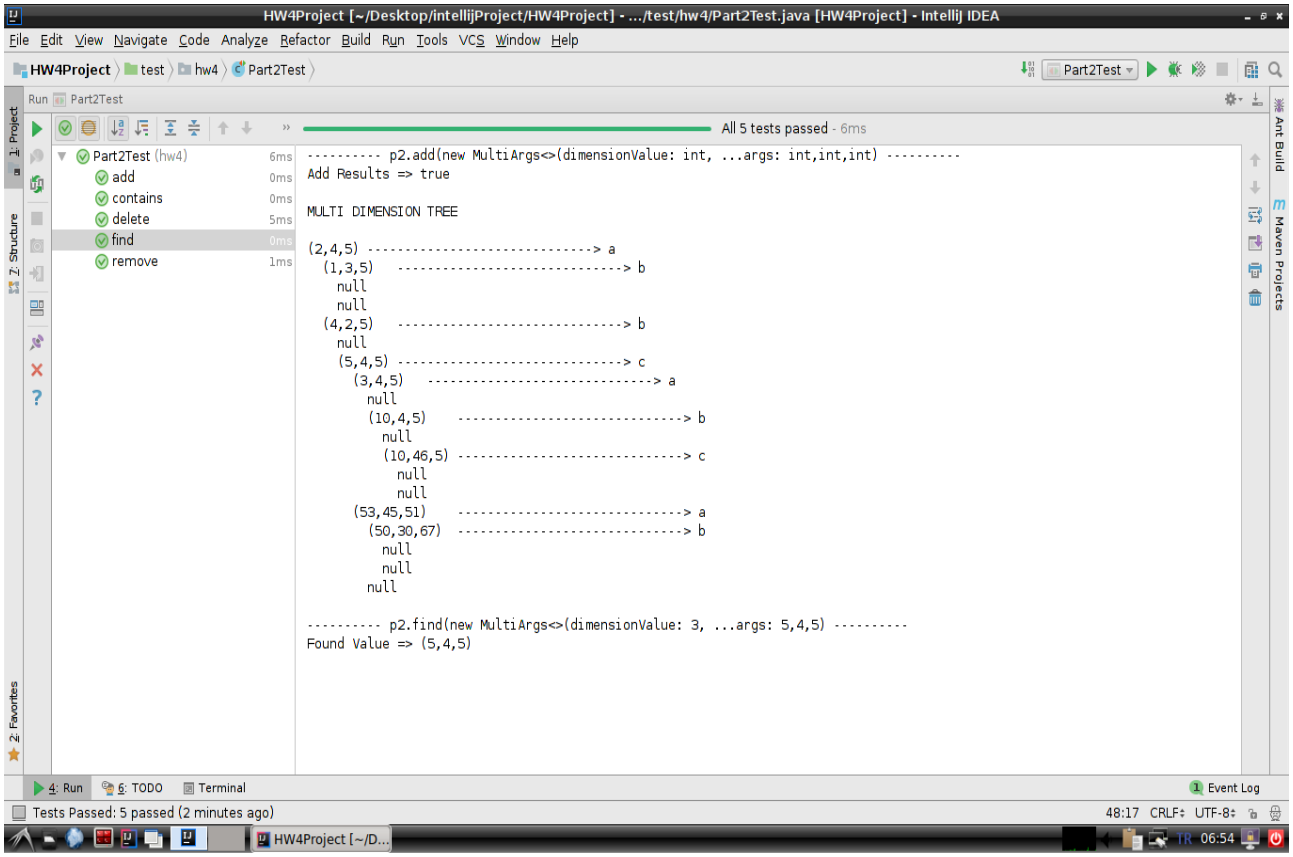


p2.delete(new MultiArgs<>(dimensionValue: 3, ...args: 3,4,5)

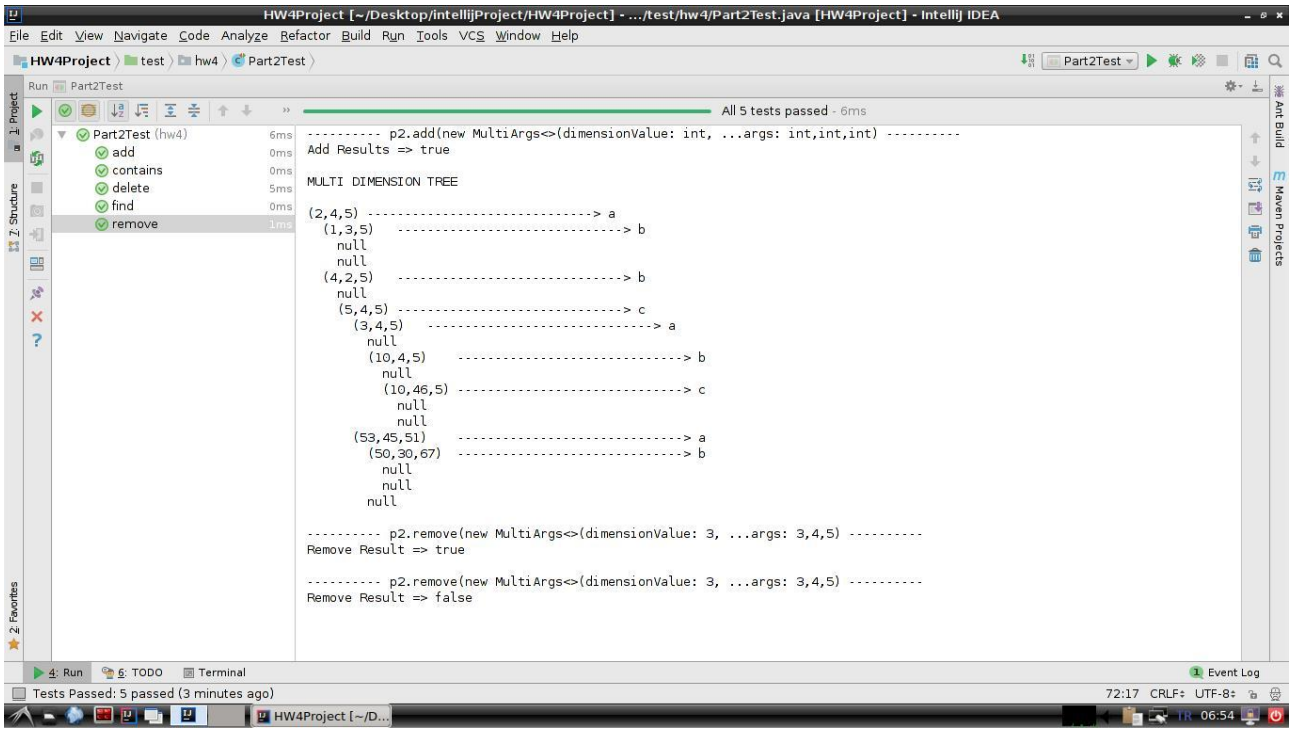


p2.find(new MultiArgs<>(dimensionValue: 3, ...args: 5,4,5)

1

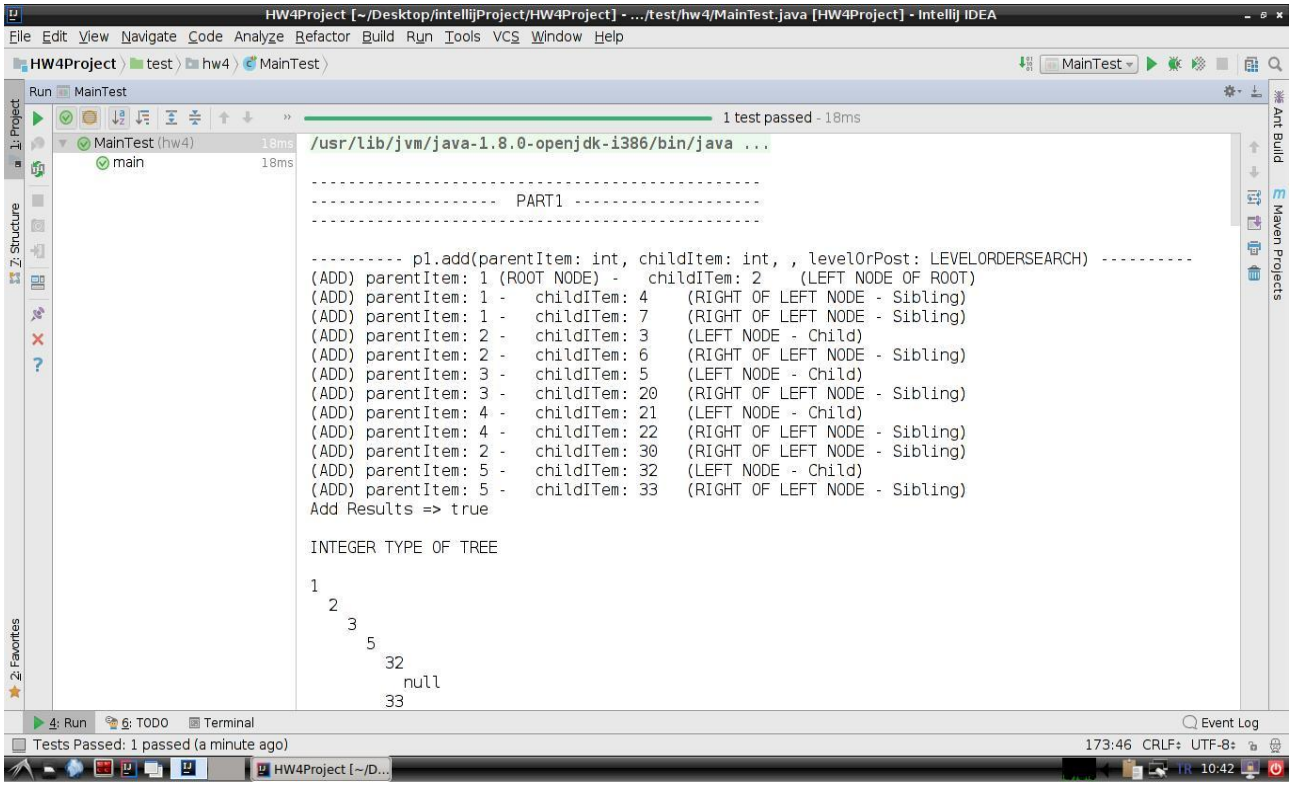


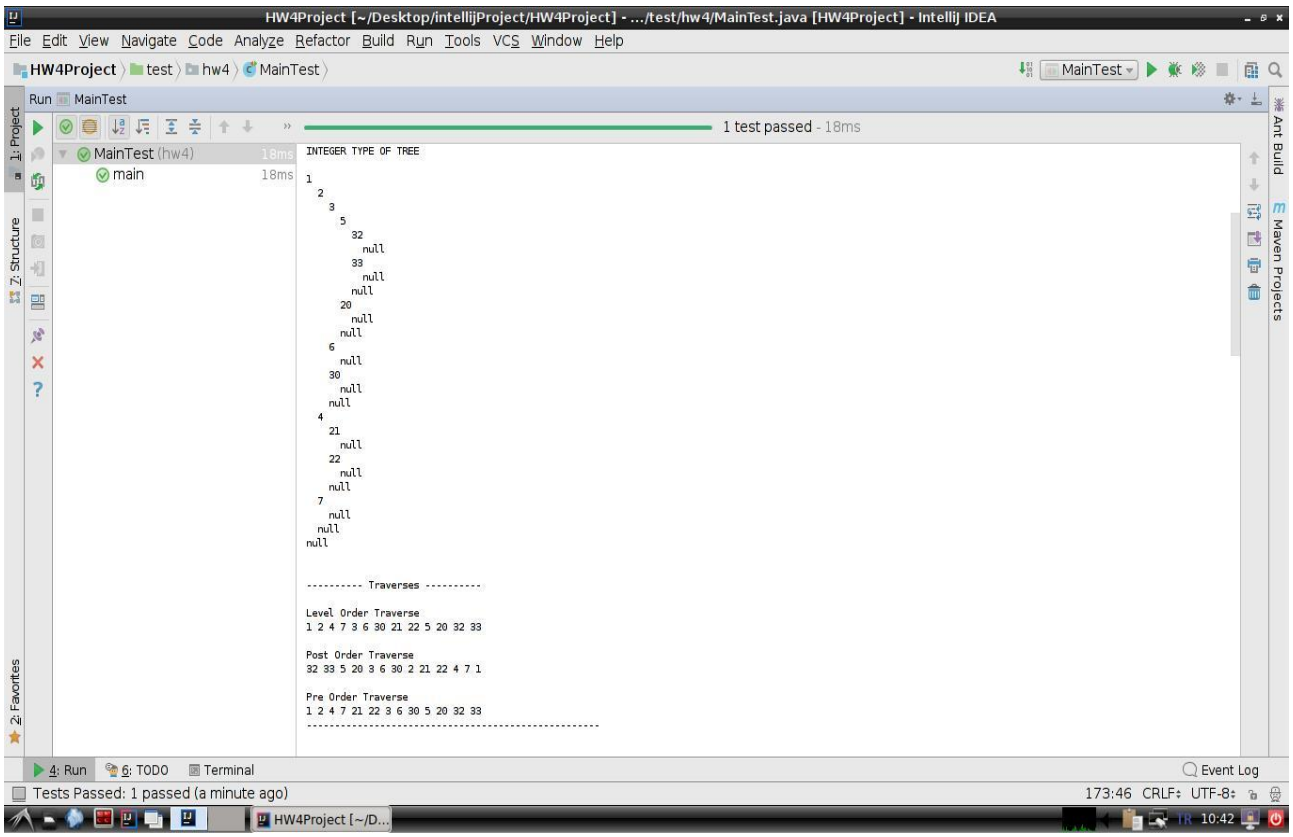
p2.remove(new MultiArgs<>(dimensionValue: 3, ...args: 3,4,5)
p2.remove(new MultiArgs<>(dimensionValue: 3, ...args: 3,4,5)

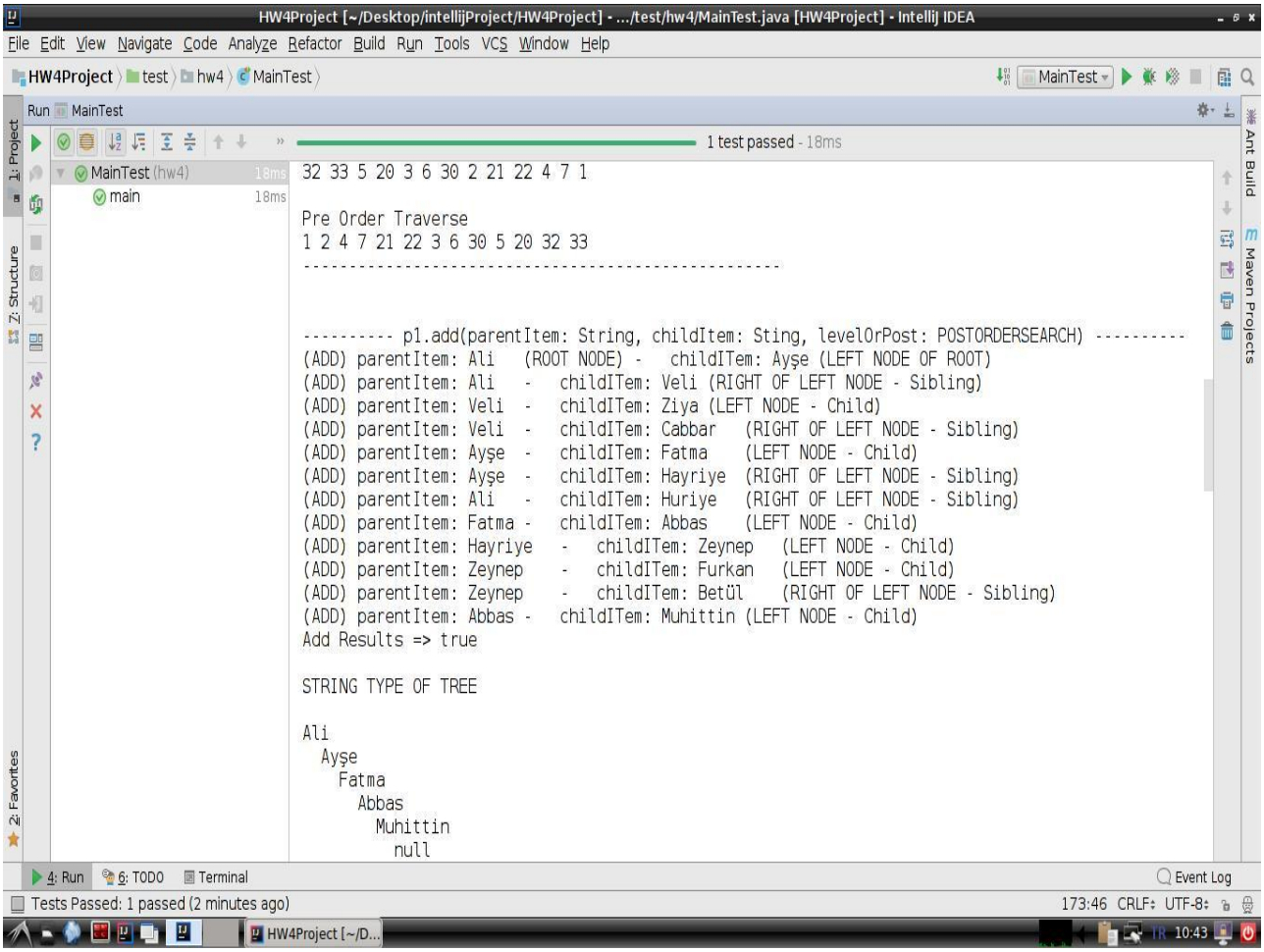


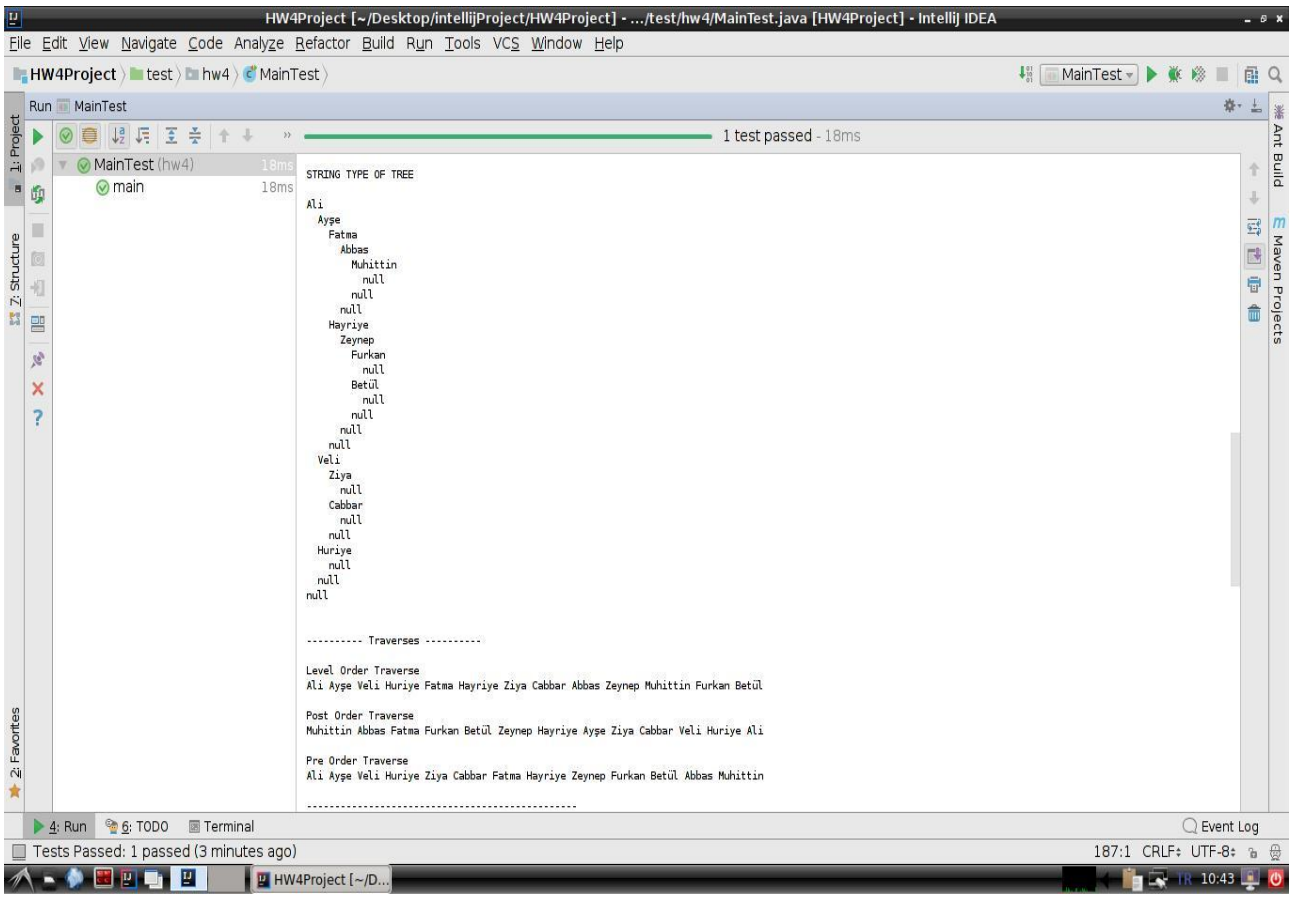
3.2 Running Results

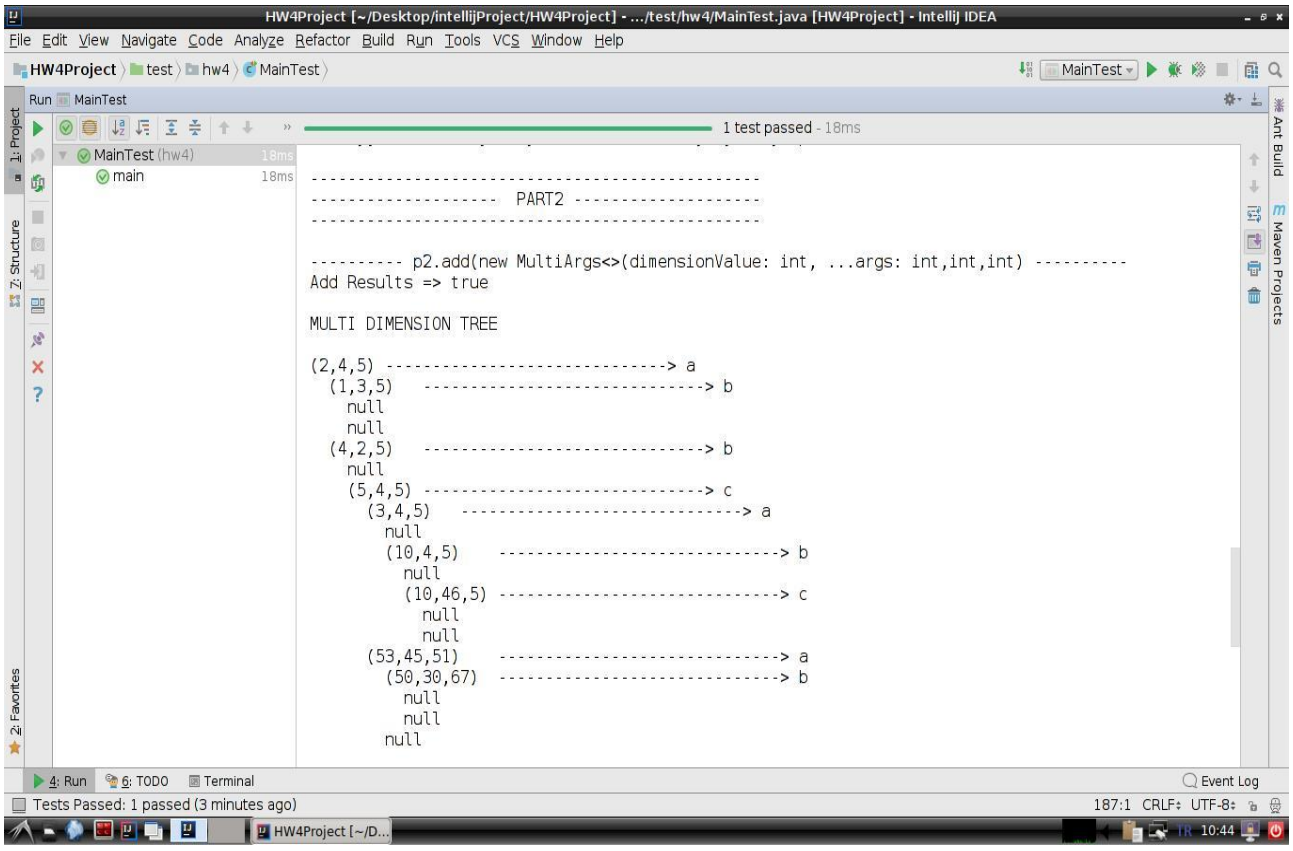
MainTest() – Result => Main içerisine yazılmış kodların testi ve ekran çıktısı.

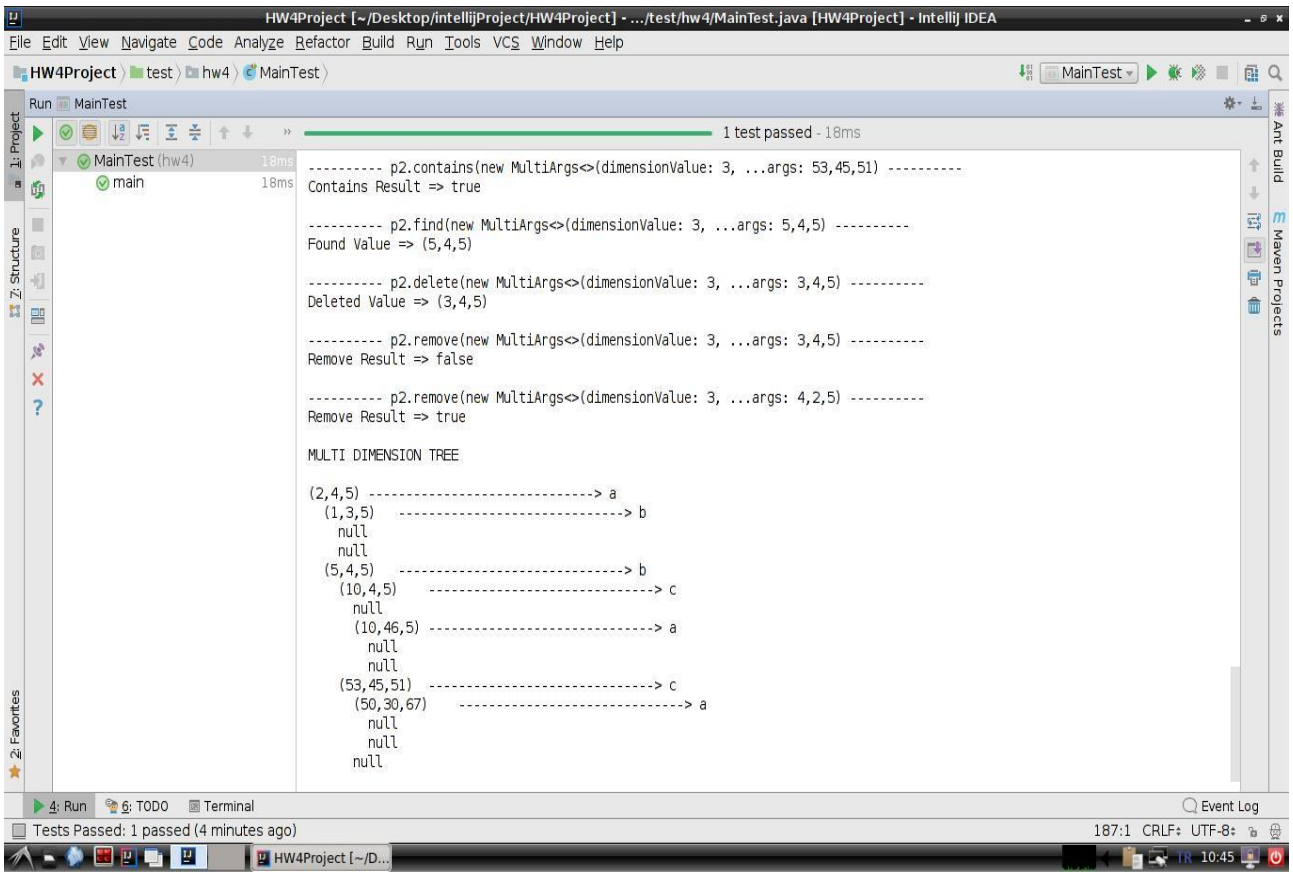



HW4Project [~/Desktop/intellijProject/HW4Project] - .../test/hw4/MainTest.java [HW4Project] - Intellij IDEA
File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
HW4Project test hw4 MainTest
Run MainTest
1 test passed - 18ms
MainTest (hw4) 18ms
main 18ms
INTEGER TYPE OF TREE
1
2
3
5
32
null
33
null
null
20
null
null
6
null
30
null
null
4
21
null
22
null
null
7
null
null
null
--------- Traverses ---------
Level Order Traverse
1 2 4 7 3 6 30 21 22 5 20 32 33
Post Order Traverse
32 33 5 20 3 6 30 2 21 22 4 7 1
Pre Order Traverse
1 2 4 7 21 22 3 6 30 5 20 32 33
4: Run 6: TODO Terminal Event Log
Tests Passed: 1 passed (a minute ago)
173:46 CRLF: UTF-8:
HW4Project [~/D...
10:42


HW4Project [~/Desktop/intellijProject/HW4Project] - .../test/hw4/MainTest.java [HW4Project] - Intellij IDEA
File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
HW4Project test hw4 MainTest
Run MainTest
1 test passed - 18ms
MainTest (hw4) 18ms
main 18ms
32 33 5 20 3 6 30 2 21 22 4 7 1
Pre Order Traverse
1 2 4 7 21 22 3 6 30 5 20 32 33
---------- p1.add(parentItem: String, childItem: Sting, levelOrPost: POSTORDERSEARCH) ----------
(ADD) parentItem: Ali (ROOT NODE) - childITem: Ayşe (LEFT NODE OF ROOT)
(ADD) parentItem: Ali - childITem: Veli (RIGHT OF LEFT NODE - Sibling)
(ADD) parentItem: Veli - childITem: Ziya (LEFT NODE - Child)
(ADD) parentItem: Veli - childITem: Cabbar (RIGHT OF LEFT NODE - Sibling)
(ADD) parentItem: Ayşe - childITem: Fatma (LEFT NODE - Child)
(ADD) parentItem: Ayşe - childITem: Hayriye (RIGHT OF LEFT NODE - Sibling)
(ADD) parentItem: Ali - childITem: Huriye (RIGHT OF LEFT NODE - Sibling)
(ADD) parentItem: Fatma - childITem: Abbas (LEFT NODE - Child)
(ADD) parentItem: Hayriye - childITem: Zeynep (LEFT NODE - Child)
(ADD) parentItem: Zeynep - childITem: Furkan (LEFT NODE - Child)
(ADD) parentItem: Zeynep - childITem: Betül (RIGHT OF LEFT NODE - Sibling)
(ADD) parentItem: Abbas - childITem: Muhittin (LEFT NODE - Child)
Add Results => true
STRING TYPE OF TREE
Ali
Ayşe
Fatma
Abbas
Muhittin
null
4: Run 6: TODO Terminal Event Log
Tests Passed: 1 passed (2 minutes ago)
173:46 CRLF: UTF-8:
HW4Project [~/D...
10:43


HW4Project [~/Desktop/intellijProject/HW4Project] - .../test/hw4/MainTest.java [HW4Project] - Intellij IDEA
File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
HW4Project test hw4 MainTest
Run MainTest
1 test passed - 18ms
MainTest (hw4) 18ms
main 18ms
STRING TYPE OF TREE
Ali
Ayşe
Fatma
Abbas
Muhittin
null
null
null
Hayriye
Zeynep
Furkan
null
Betül
null
null
null
null
Veli
Ziya
null
Cabbar
null
null
Huriye
null
null
null
--------- Traverses ----------
Level Order Traverse
Ali Ayşe Veli Huriye Fatma Hayriye Ziya Cabbar Abbas Zeynep Muhittin Furkan Betül
Post Order Traverse
Muhittin Abbas Fatma Furkan Betül Zeynep Hayriye Ayşe Ziya Cabbar Veli Huriye Ali
Pre Order Traverse
Ali Ayşe Veli Huriye Ziya Cabbar Fatma Hayriye Zeynep Furkan Betül Abbas Muhittin
4: Run 6: TODO Terminal Event Log
Tests Passed: 1 passed (3 minutes ago)
187:1 CRLF: UTF-8:
HW4Project [~/D...
10:43


HW4Project [~/Desktop/intellijProject/HW4Project] - .../test/hw4/MainTest.java [HW4Project] - Intellij IDEA
File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
HW4Project test hw4 MainTest
Run MainTest
1 test passed - 18ms
MainTest (hw4) 18ms
main 18ms
---------- PART2 ----------
---------- p2.add(new MultiArgs<>(dimensionValue: int, ...args: int,int,int) ----------
Add Results => true
MULTI DIMENSION TREE
(2,4,5) ----------> a
(1,3,5) ----------> b
null
null
(4,2,5) ----------> b
null
(5,4,5) ----------> c
(3,4,5) ----------> a
null
(10,4,5) ----------> b
null
(10,46,5) ----------> c
null
null
(53,45,51) ----------> a
(50,30,67) ----------> b
null
null
null
4: Run 6: TODO Terminal Event Log
Tests Passed: 1 passed (3 minutes ago)
187:1 CRLF: UTF-8:
HW4Project [~/D...
10:44



4 Complexity Analysis

4.1 Part1

LevelOrderSearch: Bu metotta parentItem değerine göre Binary Tree üzerinde Genral Tree gibi iterative olarak level level gezinerek arama işlemi yapılmaktadır. Metot içerisinde bulunan ilk node yani root'un null olup olmadığı kontrol edildiği için root' un null olması durumunda ya da aranan parent değerinin root olması durumunda metodun çalışması best-case de constant time Q(1) kadar sürmektedir. Ancak root da veri var ise diğer işlemlere bakılır. N children n tane parent üzerinde search ettiğimizi var sayarsak ve her bir parent ve children için queue yapısına ekleme ve daha sonra queueden silmek için bir döngü çalışacağından worst-case durumunda cubic time $O(n^3)$ kadar sürmektedir.

PostOrderSearch: Bu metotta parentItem değerine göre Binary Tree üzerinde Genral Tree gibi recursive olarak <left><right><root> gezinerek arama işlemi yapmaktadır. Metot içerisinde bulunan ilk node yani root'un null olup olmadığı kontrol edildiği için root' un null olması durumunda ya da aranan parent değerinin root olması durumunda metodun çalışması best-case de constant time Q(1) kadar sürmektedir. Ancak root da veri var ise

recurrence relation işlemi yapılır. Bu durumda da bu metot worst-case durumunda lineer time O(n) kadar sürmektedir.

PreOrderTraversal: Bu metotta Binary Tree üzerinde Genral Tree gibi recursive olarak <root> <left><right> gezinerek traverse işlemi yapmaktadır. Metot içerisinde bulunan ilk node yani root'un null olup olmadığı kontrol edildiği için root' un null olması durumunda metodun çalışması best-case de constant time Q(1) kadar sürmektedir. Ancak root da veri var ise recurrence relation işlemi yapılır. Bu durumda da bu metot worst-case durumunda lineer time O(n) kadar sürmektedir.

Add: Bu metodun hesaplanması için levelOrPost parametresinin 1 mi? 2 mi? aldığı çok önemlidir. 1 değerini aldığı takdirde içerisinde önce levelOrderSeach yapacak ve bize aranan parentItem değerinin node' unu verecek ve daha sonrasında bu node üzerinde çocuk olup olmadığı bakılır eğer çocuk yoksa worst-case durumu levelOrderSearch + while döngüsü kadar olacağından $O(n^3) + O(n) = O(n^3)$ cubic time kadar sürmektedir. Fakat 2 değerini aldığı takdirde ise içerisinde postOrderSeach işlemi yapacak ve bize aranan parentItem değerinin node' unu verecek ve daha sonrasında bu node üzerinde çocuk olup olmadığı bakılır eğer çocuk yoksa worst-case durumu postOrderSearch + while döngüsü kadar olacağından O(n) + O(n) = O(n) lineer time kadar sürmektedir.

4.2 Part2

Add: Multi Dimension Tree üzerinde recursive bir şekilde bir node' un null çocuğuna item değeri eklenileceginden ilk başta yardımcı sınıf olan MultiArgs sınıfına bakılır ve n kadar eleman içeren node yapısı üzerinde işlem yapılacağından bu node başlangıçta O(n) kadar sürer. Bu işlem her recursive çağrıda bakılacağından ve bu recursive çağrıda O(n) kadar süreceğinden worst-case durumunda quadratic time $O(n^2)$ kadar sürecektir. Best-case durumunda ise root elemani boş ise ona ekleme yapacağından ve yardımcı sınıfın çalışma süresinden dolayı lineer time O(n) kadar sürecektir.

Find: Part2 de yardımcı sınıftan kaynaklı olarak O(n) lik bir kısım her zaman çalışacaktır. Bu metotta da search işlemi recursive olarak yapılacağından ve recurrence relation hesaplaması ile her bir çağrılmada yardımcı sınıf kadar işlem yapacağından worst-case durumunda search işlemi * yardımcı sınıf = O(n) * O(n) = $O(n^2)$ quadratic time kadar

sürecekir. Best-case durumunda ise root aranılan değer olduğu var sayılırsa sadece yardımcı sınıf kadar lineer time O(n) kadar sürecektir.

Contains: Bu metot içerisinde find metodu çağrılacağından find metodundaki worst-case ve best-case durumları ile aynıdır.

Delete: Bu metoda find metodu gibi çalışmaktadır. Farklı olarak içerisinde findBiggestChildHelper metodunu çağırması ve bu metodun O(n) kadar sürmesidir. Worst-case durumunda bu yardımcı metodu da işin içine katarsak cubic time $O(n^3)$ kadar sürmektedir. Best-case de ise lineer time O(n) kadar sürecektir.

Remove: Bu metot içerisinde delete metodu çağrılacağından delete metodundaki worst-case ve best case durumları ile aynıdır.