

**Gebze Technical University
Computer Engineering**

CSE 222 - 2018 Spring

HOMEWORK 4 REPORT

**AKIN ÇAM
151044007**

Course Assistant:

1 INTRODUCTION

1.1 Problem Definition

Part1.

Genel bir ağaç yapısı oluşturulması istenmiş. Bunun için Data Structure and Algorithm kitabında bulunan BinaryTree extends edilmesi istenmiştir. Genel ağaç yapısı şu metodları içerir.

1-preOrderTraverse:

Bu method BinaryTree den override edilmiştir ve kardeşler aynı sırada parent ın çocuğu bir tab sağda olacak şekilde implement edilmiştir.

2-postOrderSearch:

Bu method ağacın ilk önce sol tarafını sonra sağ tarafını en son root u arayacak şekilde recursive olarak tasarlanacaktır.

3-levelOrderSearch:

Bu method ilk root sonra root un çocukları şeklinde seviye seviye arama gerçekleştirecektir.

4-add:

Bu method ağaca eleman ekleyecektir. Eğer eleman eklenemiyorsa true ya da false return eder.

Part2.

Bu bölümde ağacın her node un da bir eleman yerine elemanlar kümesi olan ve eleman eklenirken her sonraki seviye için node un bir sonraki elemanını ve nodeun içindeki eleman sayısına ulaşınca başa dönüp bu şekilde sırayan bir ağaç yapısı istenmiş ve şu metodları içerir.

1-add:

Bu method her levelde o levelle indexi karşılaştırarak küçük ise sola büyük ise sağa ekler. Level sayısının nodedaki eleman sayısı modunu alarak devam eder.

2-contains(bool)-find:

Ağaç içinde elemanı arar ve buna göre return değeri oluşturur.

3-delete-remove(bool):

Ağaç içinde elemanı arar ve sildikten sonra ağaçta o ilgili yerdeki elemanları yeniden düzenler.

1.2 System Requirements

Part1:

Add(): bu method parentItem alır eğer ağaç boşsa root a ekler değilse parent ı arar bulursa child ı parentın varsa kardeşlerinin yanına yoksa soluna ekler. Çalışma zamanı en iyi ihtimalle root parent olması veya rootun null olmasıdır $\Omega(1)$ dir. En kötü ihtimalle en alt

levele kadar iner çünkü levelOrderSearch kullanılır lineerdir $O(n)$ olur. Çalışma zamanına average case de $O(n)$ diyebiliriz.

LevelOrderSearch(): bu method aranmak istenen elemanı level level arar ve recursive bir methoddur. Yardımcı olarak root, queue ve ilgili node u alan bir private method yazılmıştır. Queue kullanılır çünkü queue eklenen her eleman aranacaktır ve ilk eklendiği anda aranması gerekir ve queue fifo olduğu için bu kullanılır. Çalışma zamanı için lineer olarak ilerlediği için ve for while vs loop olmadığı için $O(n)$ yani $\omega(n)$ diyebiliriz.

PostOrderSearch(): bu method aranmak istenen elemanı önce sol sonra sağ ağaç ve en son root olmak üzere aramaktadır. Yardımcı olarak root, stack ve ilgili node u alan bir private method yazılmıştır. Node return eder. Çalışma zamanı için $n \cdot O(n)$ diyebiliriz çünkü recursive olduğu için lineer hareket eder ve her if bloğunda birden çok işlem olduğu için.

PreOrderTraverse(): bu method ağacı ekrana yazar. Super classtan farklı olarak eğer sol ağaca geçince derinlik artacak sağa geçtiğinde sabit kalacaktır. Çalışma zamanı lineerdir ve $O(n)$ dir diyebiliriz.

Part2:

Bu bölümde ilk önce nodun içinde bulunacak eleman sayısı bu classın constructuna gönderilmelidir. Çünkü add delete gibi methodlar bu size ı baz alarak yapar ve farklı bir sayıya sahip olan eleman gönderildiğinde bu exception handle edilebilir olmalıdır. Inner class bir vector tutar. Gönderilen vector bu vectore clone edilir. Ve node a eklenir.

Altaki 5 method da private yardımcı metodlara sahiptir bunlar: verilen parametrenin inner class a cast edilmiş hali bir node içinde bulunan toplam eleman sayısı ve root gibi paremetreler alır.

Add(): gönderilen eleman inner class tipine cast edilir ve private add methoduna gönderilir. Bu add methodu recursive olarak verilen eleman hangi indexte ise o indexteki elemana göre karşılaştırılır ve küçük ise sola büyük ise sağa eklenir. Son size dan sonra bir sonraki eleman için mod olur ve başa döner bu index karşılaştırma işlemi. Method lineer zamanda çalışır recursivedir ve çalışma zamanına $O(n)$ diyebiliriz.

Contains()-find():

Bu methodlar ikiside eleman araması yapar. Contains bool return eder find bulunan elemanı. Contains methodu find methodunu kullanır. Find methodu elemanları her recursive call da index index karşılaştırır ve eğer eşleşen eleman sayısı ile toplam bir node

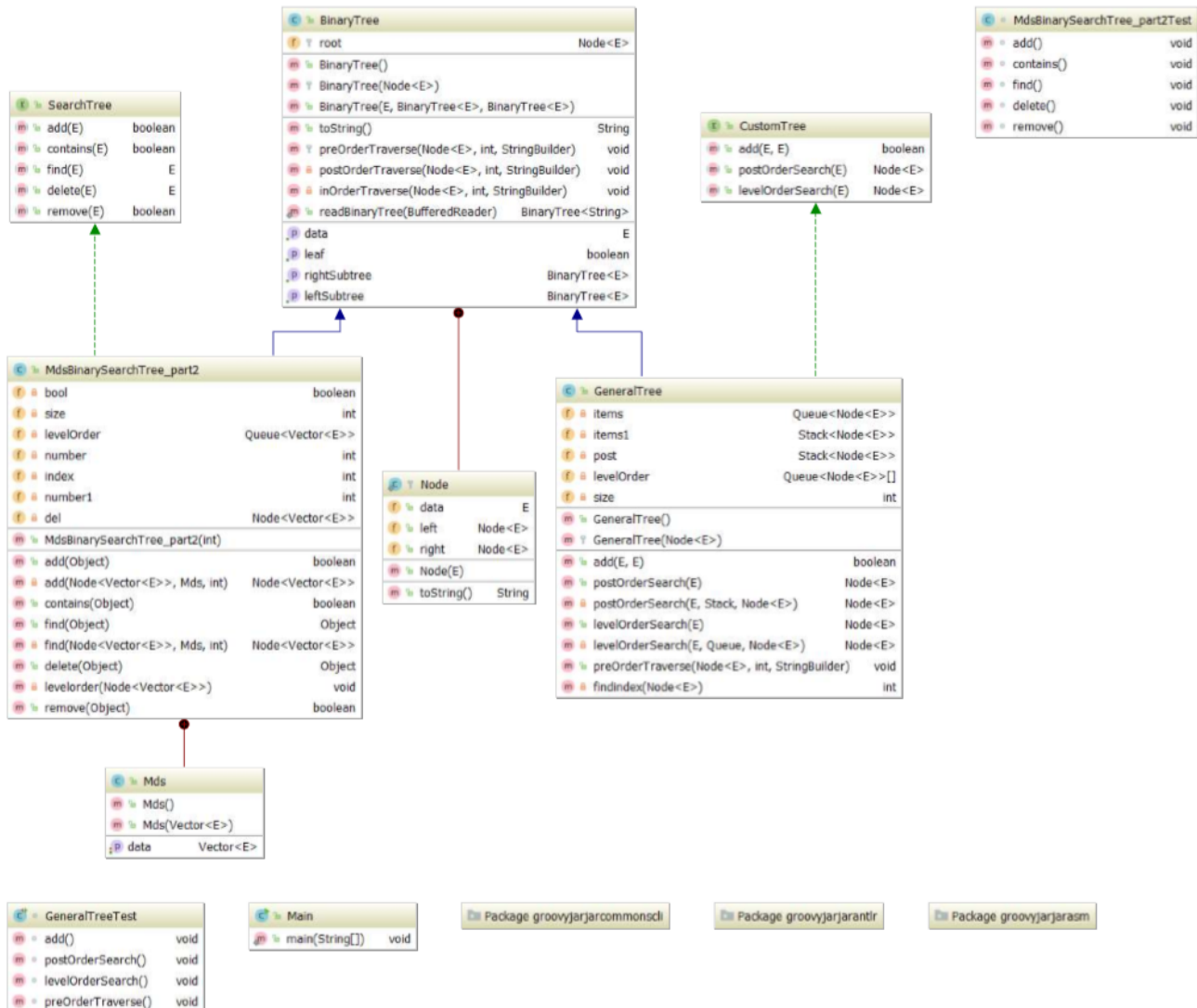
da bulunan sayı eşit ise node u return eder ve class instance variable ı true yapar. Aksi takdir de false yapar ve null return eder. Çalışma zamanı içindeki for dışında lineerdir ve $O(n)$ dir. Fakat for eleman sayısı(node içindeki) kadar çalışır ve n dir. Genel çalışma zamanı için $n \cdot O(n)$ diyebiliriz yani $O(n^2)$.

Delete-remove():

İkisinde find metodlarını kullanır elemanı bulursa remove true döndüreceğini garantiler ve delete çağrılır. Delete methodu find methodunu çağırır null değilse bütün elemanları queue ya atar. Daha sonra bu elemanları index karşılaştırarak silinmek istenen eleman hariç yeniden ağacı oluşturur. Çalışma zamanı için find methodu kullanılır $O(n^2)$. Bunun dışında kendi çalışma zamanı da levelorder n defa çalışır while n defa çalışır. $O(2n^2)$ Olur ve genel olarak $O(n^2)$ diyebiliriz.

2 METHOD

2.1 Class Diagrams



SearchTree ve CustomTree birer interfacedir.

MdsBinarySearchTree_part2 SearchTree den implements edilmiştir ve tüm metodlar implements edilmiştir. MdsBinarySearchTree_part2 BinaryTree extends edilmiştir. Mds inner classı bulunur.(is a relation) Bu class bir vector tutar ve node a bir eleman kumesi göndermeye ve orda tutmaya yardımcı olur.

BinaryTree de inner Node(is a relation) bulunur.

GeneralTree CustomTree interfacesinden implements edilmiştir. GeneralTree BinaryTree extends edildiği ve aralarında is a relation ilişkisi bulunduğu için tüm metodlarını kullanabilir(public ve protected)

2.2 Problem Solution Approach

Kitapta bulunan sizin söylediğiniz interface Search tree ve BinaryTree metodlarını oluşturdum ve kitapta verildiği şekilde yazdım.

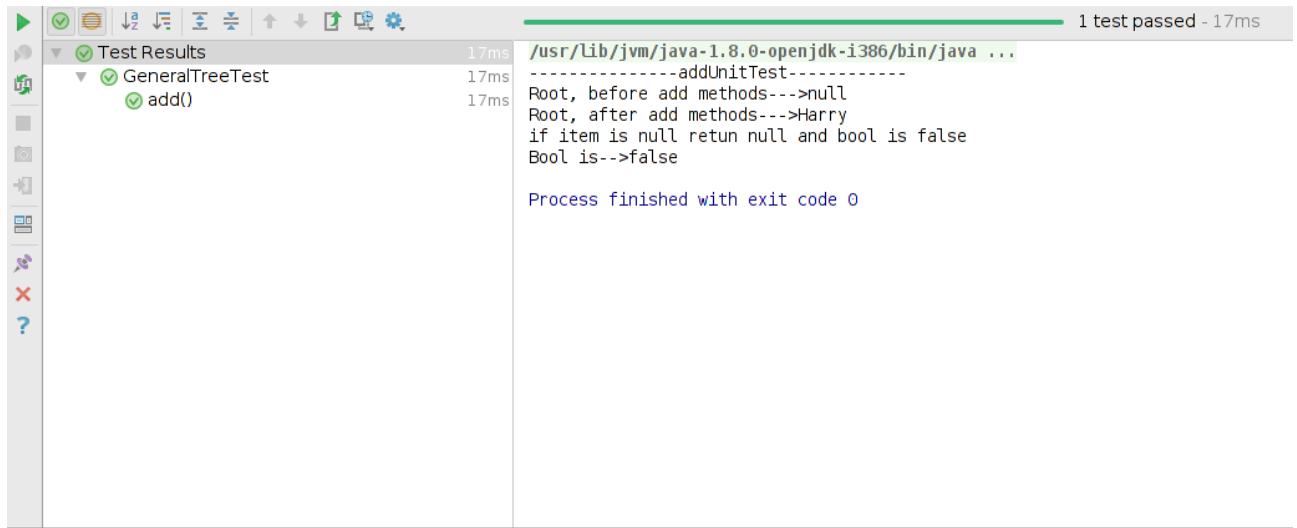
Part1 ve part2 metodları yazarken recursive metodlar kullanmaya çalıştım çünkü birbirine bağlı bir node dizisi üzerinden ilerliyor ve o node veya etrafındakilerle işlemler yapıyor. Exception handlingleri göz önüne alarak metodları implement ettim.

3 RESULT

3.1 Test Cases

Part1-UnitTest:

1-Add(): Add yaparken levelOrderSearch kullanıldığı için add yaparken üste yazanlar levellere göre üstünden geçilip aranan elemanlar.



```
Test Results 17ms /usr/lib/jvm/java-1.8.0-openjdk-i386/bin/java ...
  GeneralTreeTest 17ms
    add() 17ms
      -----addUnitTest-----
      Root, before add methods--->null
      Root, after add methods--->Harry
      if item is null return null and bool is false
      Bool is-->false

      Process finished with exit code 0
```

2-preOrderTraverse:

1 test passed - 15ms

| Test Results | 15ms | /usr/lib/jvm/java-1.8.0-openjdk-i386/bin/java ... |
|--------------------|------|--|
| GeneralTreeTest | 15ms | 1 |
| preOrderTraverse() | 15ms | 1 |
| | | 1 |
| | | 1 |
| | | 2 1 |
| | | 2 4 7 3 1 |
| | | 2 When you use add method levelorderSearch is calledso above the items print |
| | | -----preOrderTraverseUnitTest----- |
| | | result is----> |
| | | 1 |
| | | 2 |
| | | 3 |
| | | 5 |
| | | ----- |
| | | 6 |
| | | ----- |
| | | 4 |
| | | ----- |
| | | 7 |
| | | ----- |
| | | ----- |

Process finished with exit code 0

Loaded classes are up to date. Nothing to reload.

3-postOrderSearchTest:

1 test passed - 43ms

| Test Results | 43ms | /usr/lib/jvm/java-1.8.0-openjdk-i386/bin/java ... |
|-------------------|------|---|
| GeneralTreeTest | 43ms | -----postOrderSearchUnitTest----- |
| postOrderSearch() | 43ms | william1 |
| | | william1 |
| | | william1 |
| | | william1 |
| | | robert williamc |
| | | robert |
| | | william2 |
| | | adele |
| | | PostorderResult should be root(william1) |
| | | PostorderSearch return value--->william1 |
| | | Process finished with exit code 0 |
| | | |

4-levelOrderSearch:

Test Results

- GeneralTreeTest
 - levelOrderSearch() 21ms

```

/usr/lib/jvm/java-1.8.0-openjdk-i386/bin/java ...
william1
william1
william1
william1

robert When you use add method levelOrderSearch is calledso above the items print
-----levelOrderSearchUnitTest-----
levelOrderSearch items orders should be below
william1
robert
william2
adele
-----testResult-----
william1

robert william2 adele searched item adele findned item is--->adele

Process finished with exit code 0

```

Loaded classes are up to date. Nothing to reload.

PART2 UNIT TEST:

Test Results

- MdsBinarySearchTree_part2Test 34ms
 - delete() 21ms
 - remove() 7ms
 - contains() 1ms
 - add() 2ms
 - find() 3ms

```

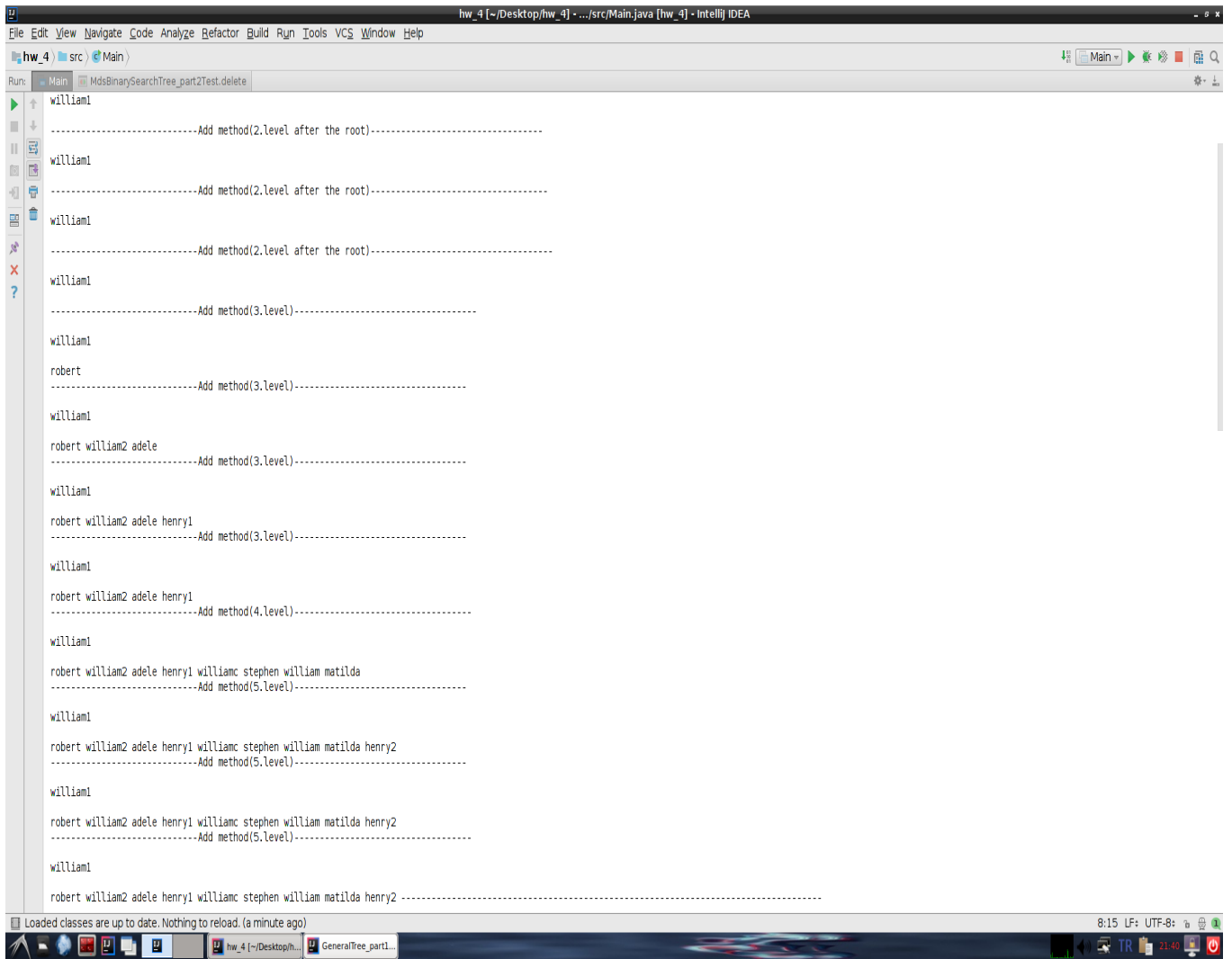
/usr/lib/jvm/java-1.8.0-openjdk-i386/bin/java ...
-----deleteUnitTest-----
num3(322,454,566,3474) is wanted to delete. Test result is--->
[45, 55, 4522, 5534]
[3, 4, 5, 344]
[451, 30, 23, 24]
[322, 454, 566, 3474]
-----
[45, 55, 4522, 5534]
[3, 4, 5, 344]
[322, 454, 566, 3474]
null
-----removeUnitTest-----
num3(322,454,566,3474) is wanted to remove. Test result is--->
false
true
-----containsUnitTest-----
num1(451,30) is looked whether if contains..
return value--->true
num2(3,6) is looked whether if contains..
num2 isnt it in tree so returns false
return value--->false
-----addUnitTest-----
if item isn null adds and return true-->test1 +mds.add(num1)-->true
if items is null return false test2-->+mds.add(null)--> false
-----findUnitTest-----
[45, 55] is searched result is-->
findned item is-->[45, 55]
searched item is is bigger than in tree items--> so throws exception
Exception is throwned----> Input size not equal other items(find method)

Process finished with exit code 0

```

3.2 Running Results

1--Part1:



-----PreOrderTraverse Test-----

```
william1
  robert
    williamc
      ----
    ----
  william2
    ----
  adele
    stephen
      ----
    ----
  henry1
    william
      ----
    matilda
      henry2
        henry
          ----
        richard1
          ----
        geoffrey
          ----
      ----
    ----
  ----
----
```

-----PostOrderSearch Test-----

first item=william1 is searched---->

```
williamc
robert
william2
stephen
adele
william
henry
richard1
geoffrey
henry2
matilda
henry1
```

william1 returns postOrderSearch

```

-----LevelOrderSearch Test-----
william1

robert william2 adele henry1

henry1  returns levelOrderSearch
-----
william1

robert william2 adele henry1 williamc stephen william matilda henry2 henry richard1

richard1  returns levelOrderSearch
-----Part1-Test2-----

-----Add method-----

1
1
1
1

2 1

2 1

2 4 7 3

-----PostOrderSearch Test-----

first item=1 is searched---->
5
3
6
2
4
7
1
-----LevelOrderSearch Test-----

1

2 4 7 3 6 5
Process finished with exit code 0

```

2-Part2 Test:

-----Part2-Test-----

```
Add method.
root-->[3, 6, 1, 8]
root.right-->[33, 61, 12, 81]
root.right.left-->[4, 5, 61, 7]
root.right.right-->[323, 64, 14, 86]
root.left-->[2, 64444, 12, 81]
root.right.right.right-->[36, 67, 18, 82]
root.right.left.left-->[333, 60, 15, 87]
root.right.left.left.left-->[32, 26, 11, 83]
root.left.left-->[2, 675, 31, 78]
-----containsTest-----
num1(451,30) is looked whether if contains..
Element is finded
-----findTest-----
[2, 675, 31, 78] is try to find. result is-->[2, 675, 31, 78]
-----deleteTest-----
root.left.left-->[2, 675, 31, 78]
root.left.left-->null
-----removeTest-----
root.left.left-->null
[2, 675, 31, 78] is searched result is-->false
-----Exception test----item number(5) is bigger than in node(4)----

Input size not equal other items(find method)

Process finished with exit code 0
```
