

Gebze Technical University
Computer Engineering

CSE 222
2017 Spring

HOMEWORK 08 REPORT

Büşra ARSLAN
131044021

Course Assistant:Şeyma Yücer

TABLE OF CONTENTS

1)Problem Solutions Approach.....

1.1) Part 1 Solution.....

1.2) Part 2 Solution.....

1.3) Part 3 Solution.....

2)Test Cases

3) Running command and results

1. Problem Solutions Approach

1.1) Part 1 Solution

Bu partta Avl tree ye eleman ekleme ve çıkarma işlemleri gerçekleştirildi.

Eleman eklerken her eleman ekledikten sonra balance durumu kontrol edildi.

Left-left parent -2 , child -1

Left-right parent -2, child +1

Right-right parent +2,child +1

Right-left parent +2, child -1 durumlarında balanced durumu bozulmuş ve left-right rotation yapılması gerekir.Her parttan sonra gerekli ise rotation edildi.

Edille,dakik ve ferc stringlerini de digerleri gibi balance durumu kontrol edilerek ekledim.

Bu işlemlerin hepsi yapıldıktan sonra Delete işleminde 1 root delete edilip balance durumu kontrol edildi , değilse rotation işlemleri gerçekleştirildi. Sonra yeni oluşan AVLTree'de yeni rootu delete ettim. Burda da balance durumu check edildi ve değilse rotation edildi. 3. Delete işlmeinde yeni root delete edildi ve diğerlerinde olduğu gibi burda da balance durumu kontrol edildi ve değilse rotation işlmei gerçekleştirildi.

Not: Nush ilk kelimesi büyük harfle başladığı için Ascii karşılığı olarak tüm küçük harflerden küçük kabul edilerek AVLTree oluşturulmuştur.

1.2) Part 2 Solution

Bu partta Skip-List yapısını anlayarak ideal skip list yapısını oluşturdum.

Bu partta 16 tane eleman vardı skip liste eklenecek. Bu 16 sayı için skip-list properties durumları levellere göre değişiklik gösterir.

16→8→4→2→1 bu şekilde level propertiesleri belirlendi. Yani;

Level 1 >> %100

Level 2 >>%50

Level 3 >> %25

Level 4 >> %12.5

Level 5 >> %6.25 oranların azalması ile nodelardaki elemanların birbirlerine bağlantıları oluşturuldu. Test Case kısmında sonuçları görebilirsiniz.

1.3) Part 3 Solution

Bu partta bizden AVL Tree'nin delete methodları implement edilerek AVL Tree test işlemleri gerçekleştirildi. Delete fonksiyonu için iki fonksiyon yazıldı. Biri private method , parametre olarak AVLNode tipinde bir node ve E tipinde bir değişken alınmıştır. Kitabın remove methodlarından faydalanılmıştır. Diğer delete methodu E tipinde bir değişken return etmiştir ve E tipinde bir parametre alınmıştır.

Classların birbirini extends veya implements ettiği classlar kitabın kodları kullanılarak projeye eklendi. Bunlar;

Search Tree (interface)

Binary Tree

BinarySearch Tree

BinarySearchTreeWithRotation ve asıl oluşturmamız gereken AVLTree kitabın kodları ile run edildi. Burda delete fonksiyonları eklendi.

Ağacın yüksekliğinin artış yada azalışına göre rotation işlemlerinin gerçekleşmesi için decrease ve increase private data field değerleri tutulmuş ve implementasyon sırasında bunların true yani artış veya azalış var durumlarına göre check işlemleri gerçekleştiriliyor.

AVLTree için yazdığım test mainde kağıda yazılı olan ödevin add işlemleri yapılarak check edilmiştir. Root remove işlemi yapılarak delete methodlarımda test edilmiştir.

String , Integer ve Character ile testler yapılmıştır.

2) Test Cases

Part 1

HW 8

Büşra ARSLAN
171044021

91) "Nush ile uslanmayani etmeli tekdir, tekdir ile uslanmayani hakki kolektir."

- insert \Rightarrow "edille", "dokik", "ferc"
- Delete \Rightarrow insert işlemlerinden sonra rootlar (3 tan) silinip her bir root silindikten sonra balance durumuna getirilmiştir.

Add "Nush" \Rightarrow

Add "ile" \Rightarrow

Add "uslanmayani" \Rightarrow

Not \Rightarrow AVL Tree'ler Binary Search Tree yapısını olduğu için aynı eleman tekrar treeye eklenmez.

Right-right (parent +2, right child +1)
 \rightarrow Denge durumu için parent left rotation yapılır. \checkmark

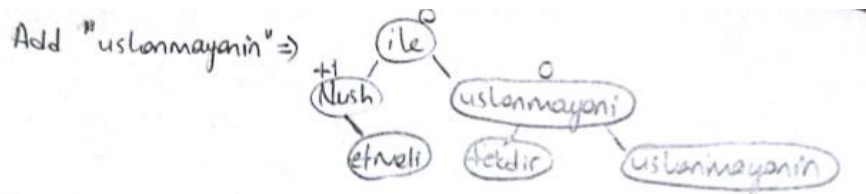
\Rightarrow Balanced sağlandı.

Add "etmeli" \Rightarrow

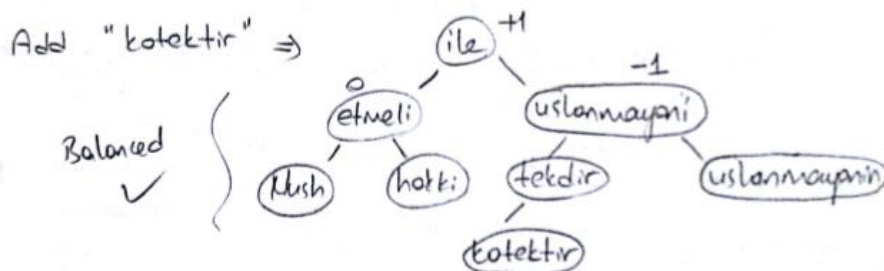
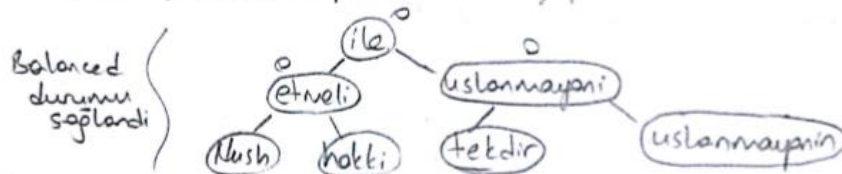
} Balanced \checkmark

Add "tekdir" \Rightarrow

} Balanced \checkmark

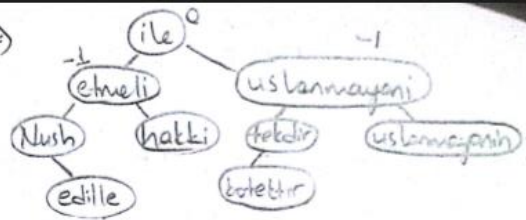


+2, +1 ⇒ Parent left rotation yapılır. ✓

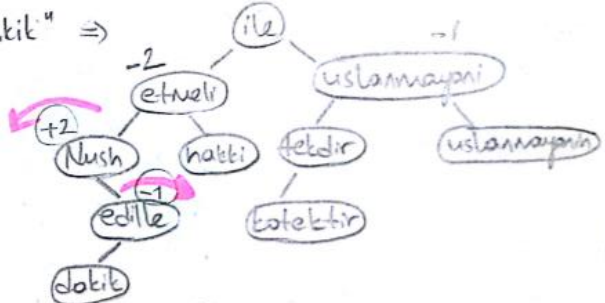


- insert "edille" \Rightarrow

Balance ✓



- insert "dokik" \Rightarrow



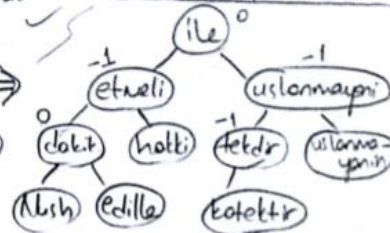
Right-left (parent $+2$, child -1)
 \rightarrow Denge durumu için child etrafında right rotation
 parent etrafında left rotation yapılır.

Child right rotation



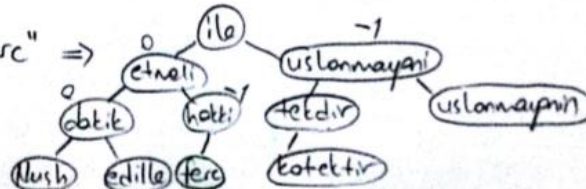
Balance ✓

Parent left rotation



- insert "ferc" \Rightarrow

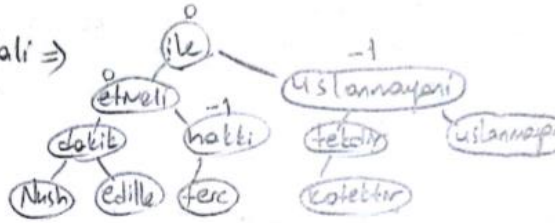
Balance ✓



Insert işlemlerinden sonra rootla silinecek diğer root silindikten sonra balanced durumuna getirilecek AVL tree. ✓

Agacın Son Hali \Rightarrow

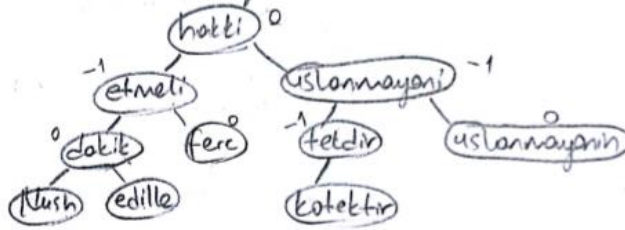
Bu an
balance
durumunda



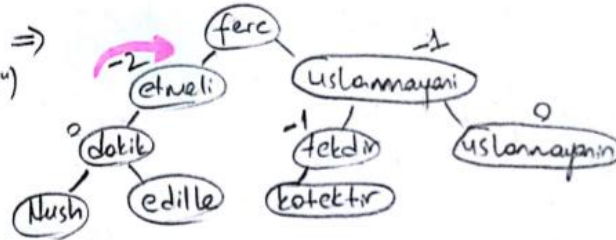
✓ Delete root
("ile") \Rightarrow

Bu durumda inorder predecessor:
root'un sol agacının en büyük elemanı
yeni root olarak seçilir.

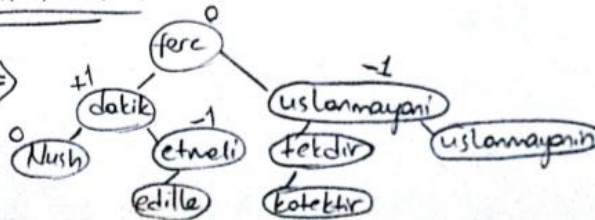
balanced
✓



✓ Delete root
("hatti") \Rightarrow



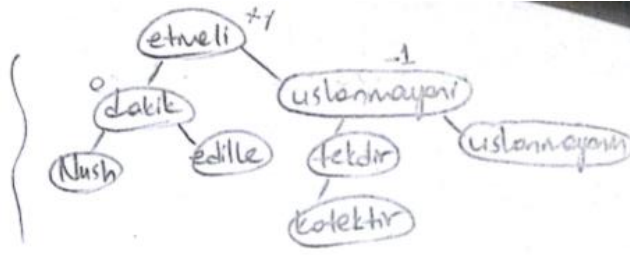
Right rotation



delete root

("fere")

Balanced



Balance bozulma durumu; (ve nasıl düzelteceği)

- ⇒ Left-left (parent -2, ^{left}child -1)
 - Parent etrafında right rotation ile balance sağlanır
- ⇒ Left-right (parent -2, left child +1)
 - Child etrafında left rotation ve parent etrafında right rotation
- ⇒ Right-right (parent +2, right child +1)
 - Parent etrafında left rotation ile balance sağlanır.
- ⇒ Right-left (parent +2, right child -1)
 - Child etrafında right rotation, parent etrafında left rotation

Balance durumunu sağlamanın amacı $O(n)$ çalışma zamanını $O(\log n)$ yapmaktır. ✓

+1, -1, 0 denge bozulma durumuna şu şekilde bakılır. Bir node'un

sağ tree level

✓ sol tree level

Denge bozulduğunda

right rotation ve

left rotation

yapılarak balance ✓

durumuna getirmeye çalışılır.

bite balanced durumunu verir.

Part 2

Ö2) İdeal skip-list

5-10-15-20-25-30-36-42-45-50-55-60-68-72-86-93

Toplam 16 eleman var.

Skiplist'teki mantık şu şekilde;

Level 1'de %100 probability

Level 2'de %50

Level 3'de %25

bu şekilde
gider.

Burada 16 eleman var.

Level 1 için %100 → 16

Level 2 için %50 → 8

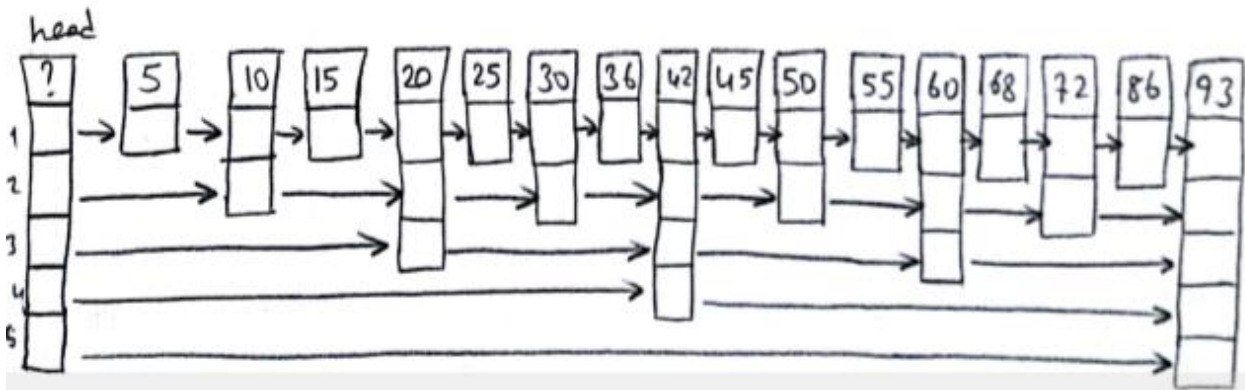
Level 3 için %25 → 4

Level 4 için %12,5 → 2

Level 5 için %6,25 → 1

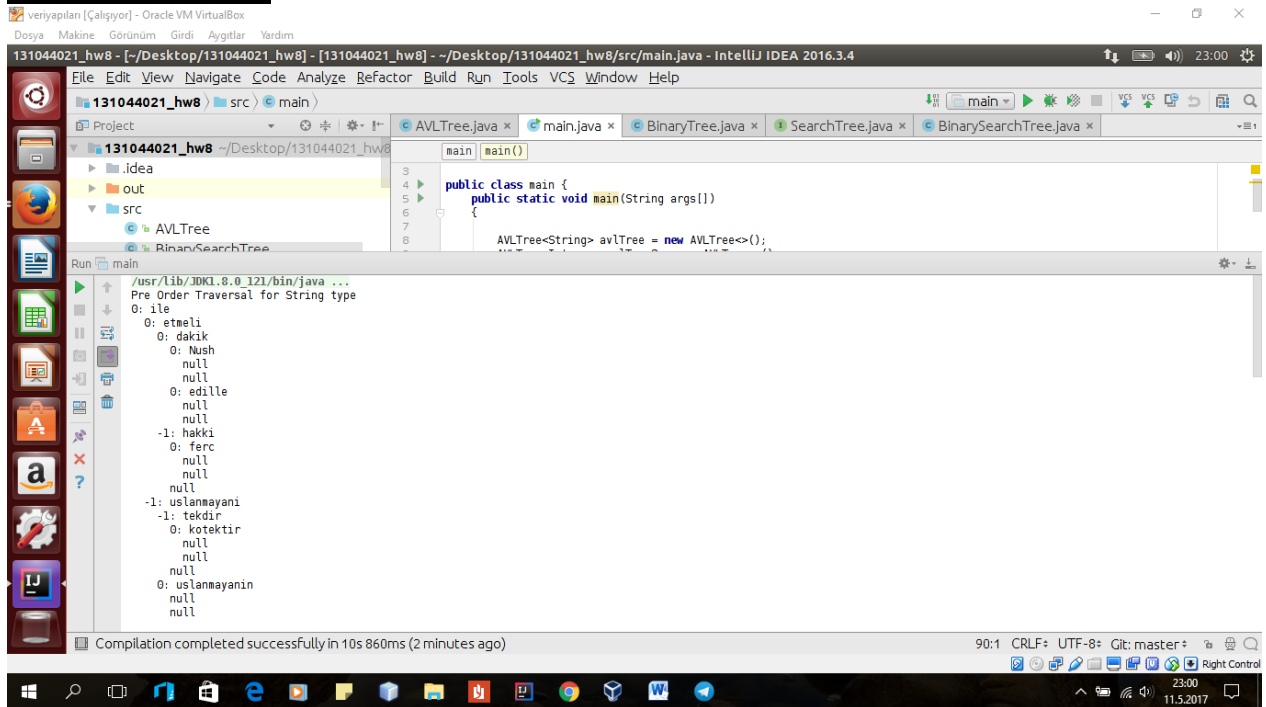
Bir node'da
maximum
5 node
olabilir.

⇒ Olasılıklarla node sayıları orantılı ise ideal skip-list denir

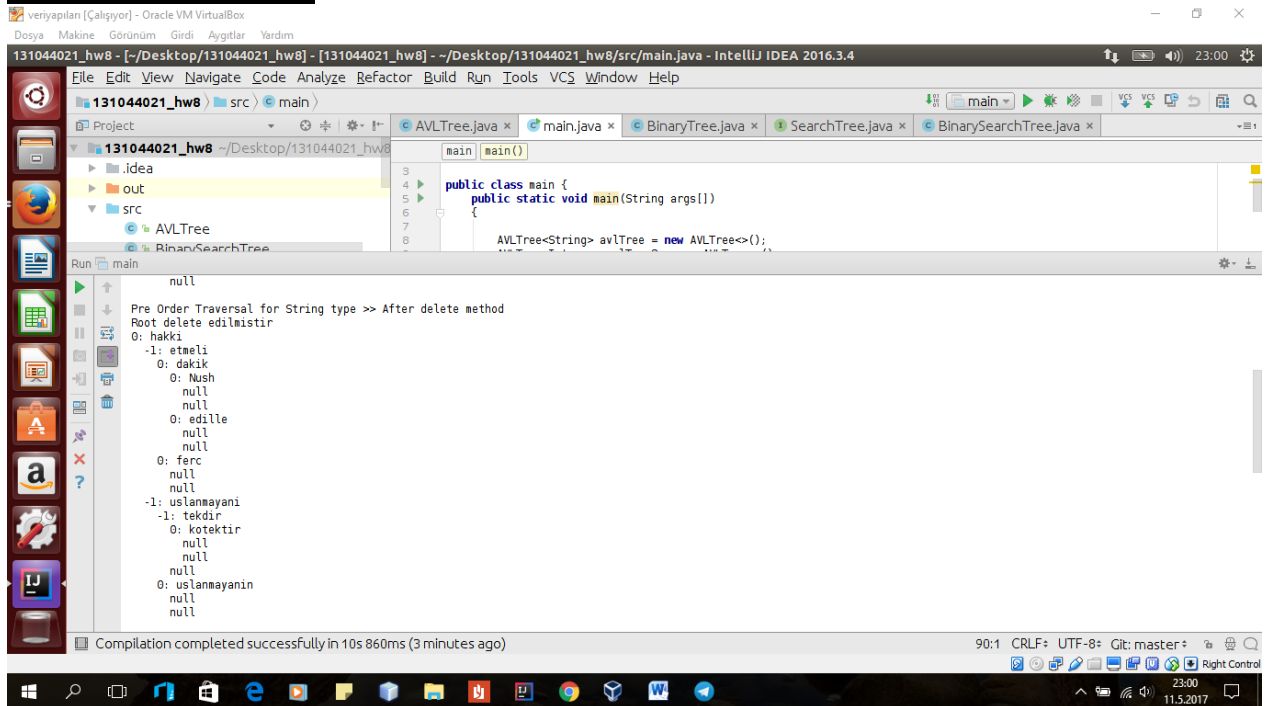


Part 3

Add method test



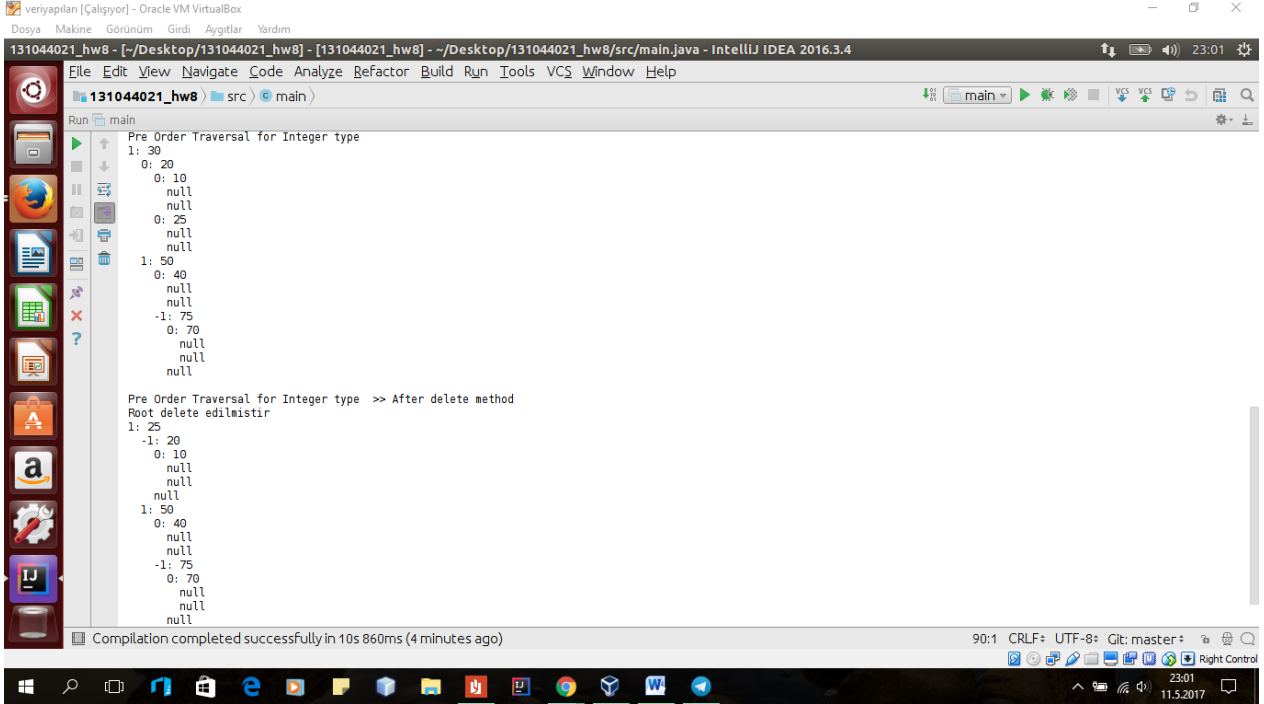
Delete Method Test



3) Running command and results

Bu bölüm part 3 için geçerlidir. Part 1 ve Part 2 nin açıklama kısmı Problem Solution Approach kısmındadır. Sonuçları da Test Case kısmında mevcuttur.

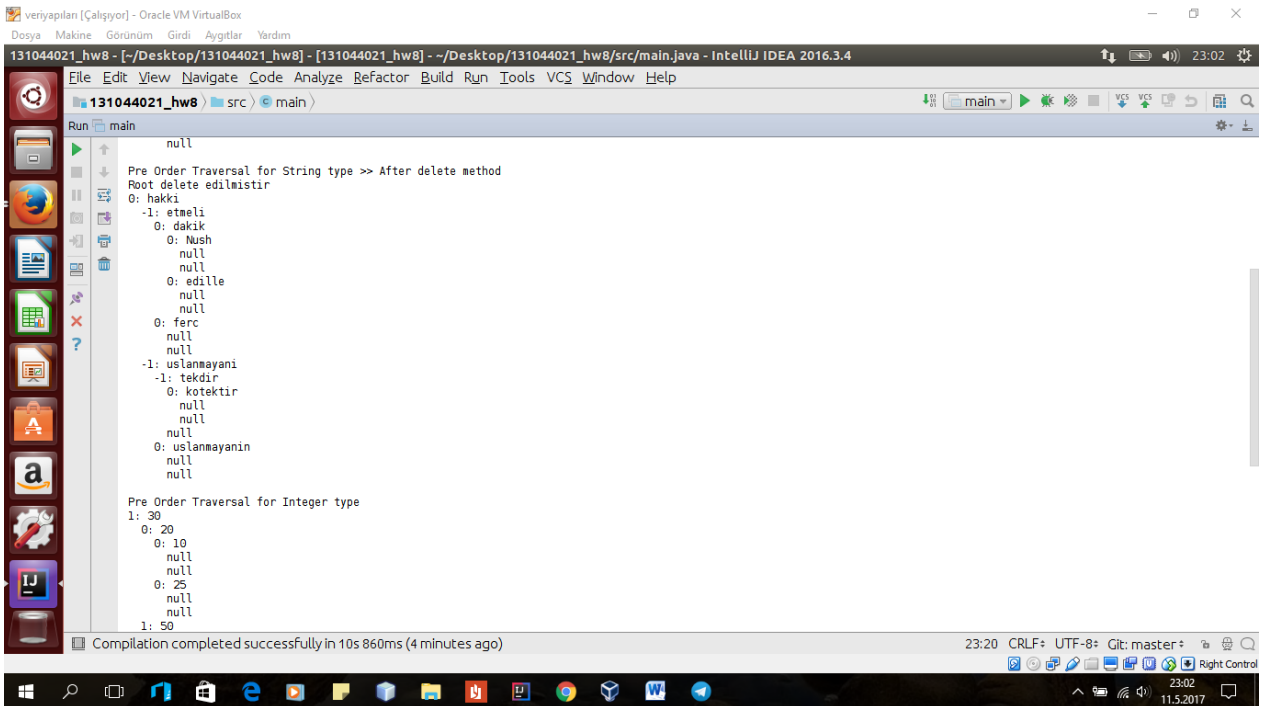
Part3 de String ve Integer ile test edilmiştir.



```
Run main
Pre Order Traversal for Integer type
1: 30
0: 20
0: 10
null
null
0: 25
null
null
1: 50
0: 40
null
null
-1: 75
0: 70
null
null
null

Pre Order Traversal for Integer type >> After delete method
Root delete edilmiştir
1: 25
-1: 20
0: 10
null
null
null
1: 50
0: 40
null
null
-1: 75
0: 70
null
null
null

Compilation completed successfully in 10s 860ms (4 minutes ago)
```



```
Run main
null

Pre Order Traversal for String type >> After delete method
Root delete edilmiştir
0: hakki
-1: etaeli
0: Nush
null
null
0: edille
null
null
0: ferc
null
null
-1: uslanmayani
-1: tekdir
0: kotektir
null
null
0: uslanmayanin
null
null

Pre Order Traversal for Integer type
1: 30
0: 20
0: 10
null
null
0: 25
null
null
1: 50

Compilation completed successfully in 10s 860ms (4 minutes ago)
```