

**Gebze Technical University
Computer Engineering**

CSE 222 - 2018 Spring

HOMEWORK 4 REPORT

**Furkan AKTAŞ
141044029**

Course Assistant: Mehmet Burak KOCA

1 INTRODUCTION

1.1 Problem Definition

Part 1: BinaryTree den extend edilmek şartıyla kitaptaki aile ağacının implemntasyonu istendi.Sola çocuklar sağa kardeşler eklenecek şekilde yapı levelOrder ve postOrder binaryTree üzerinde fakat aile tablosuna göre hareket ederek yapıldı.

Part 2: Birden fazla parametre ile çalışabilecek şekilde her levelinde farklı parametre(düzlem) ile karşılaştırarak SearchTree implement eden bir tree yapısı istendi.

1.2 System Requirements

Part 1:

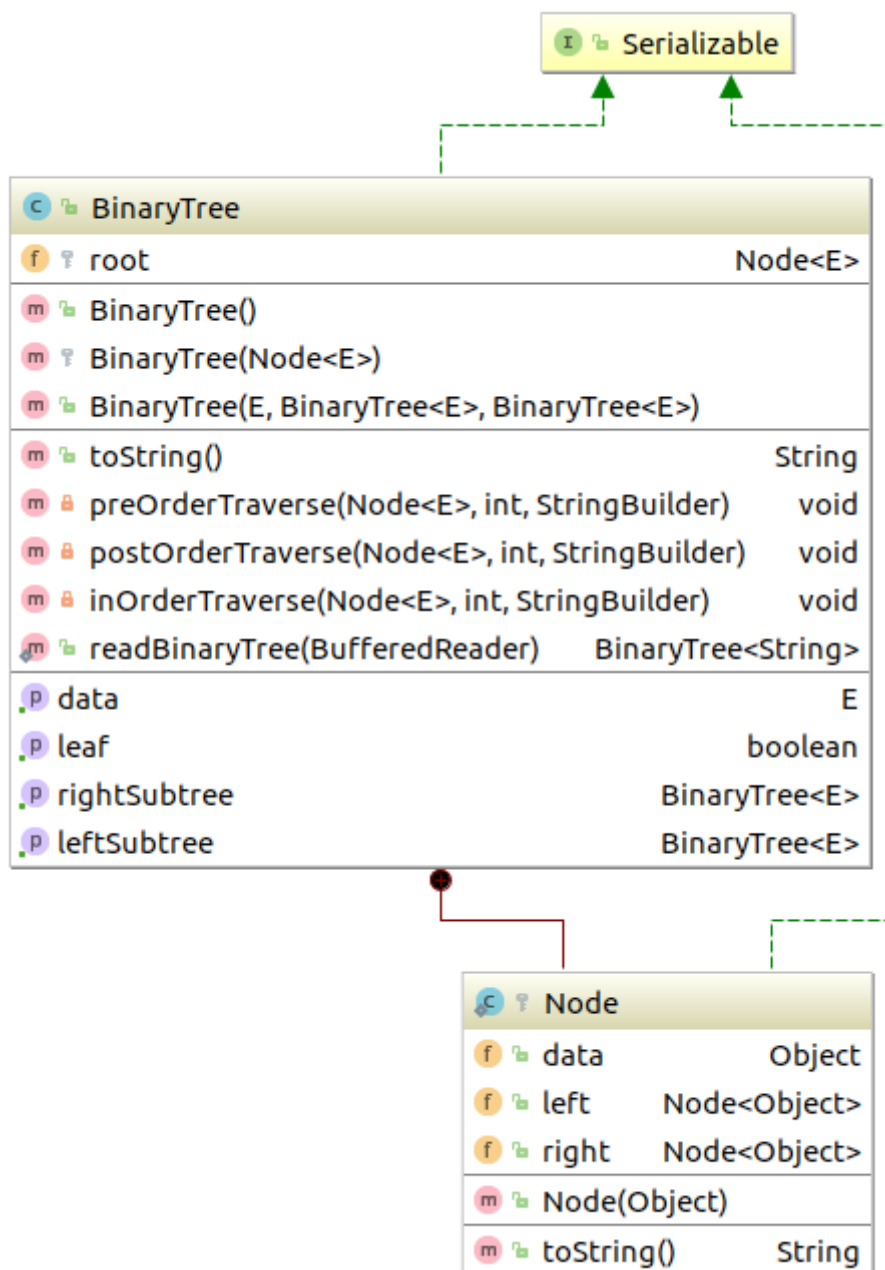
- postOrder recursive , levelOrder Queue yapısı ile yapıldı.
- preOrderTraverse, hem asıl tabloda hemde binaryTree şeklinde olarak aynı sonuçları verdiğiinden binaryTree içindeki aynı method kullanıldı.

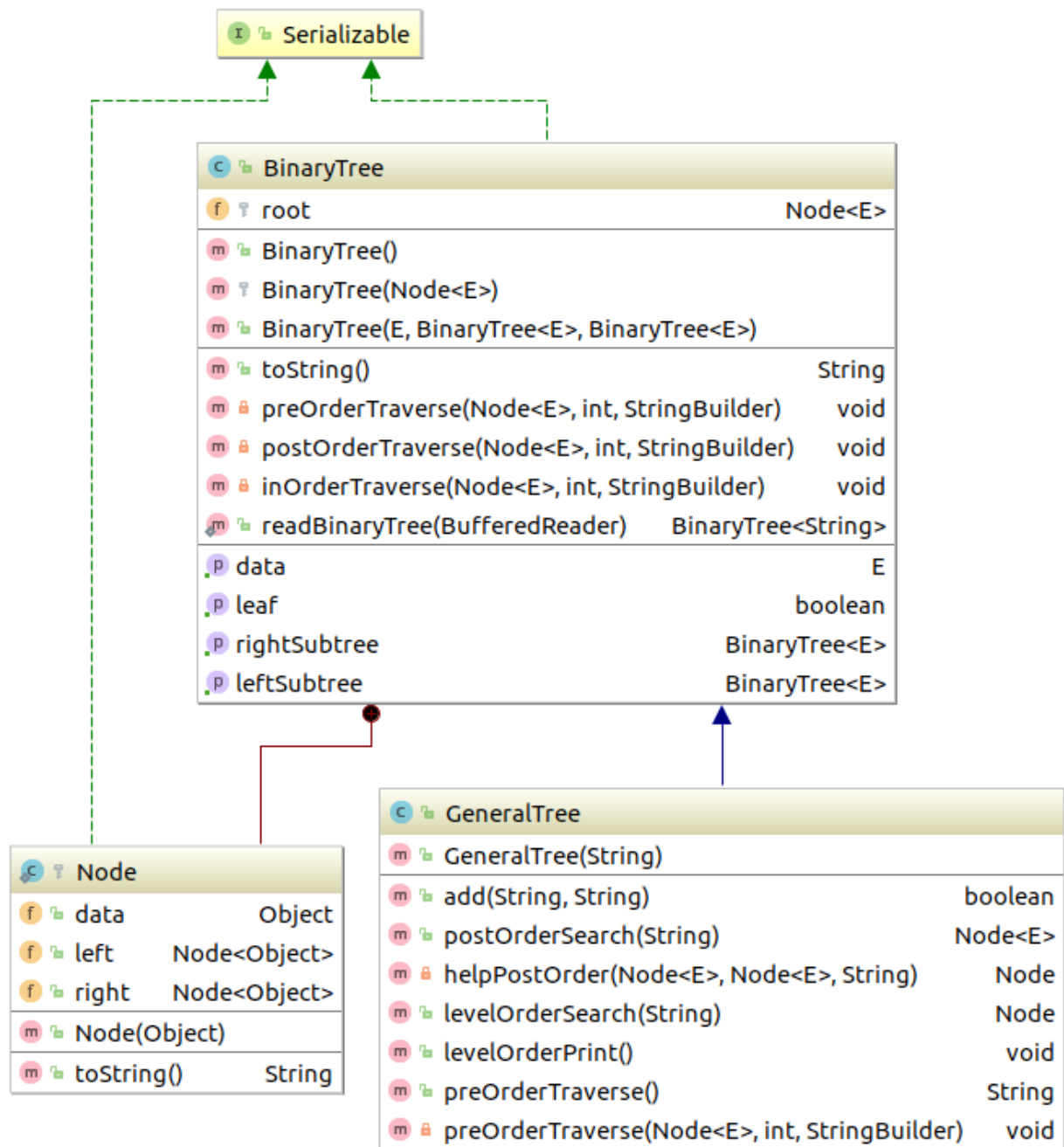
Part 2:

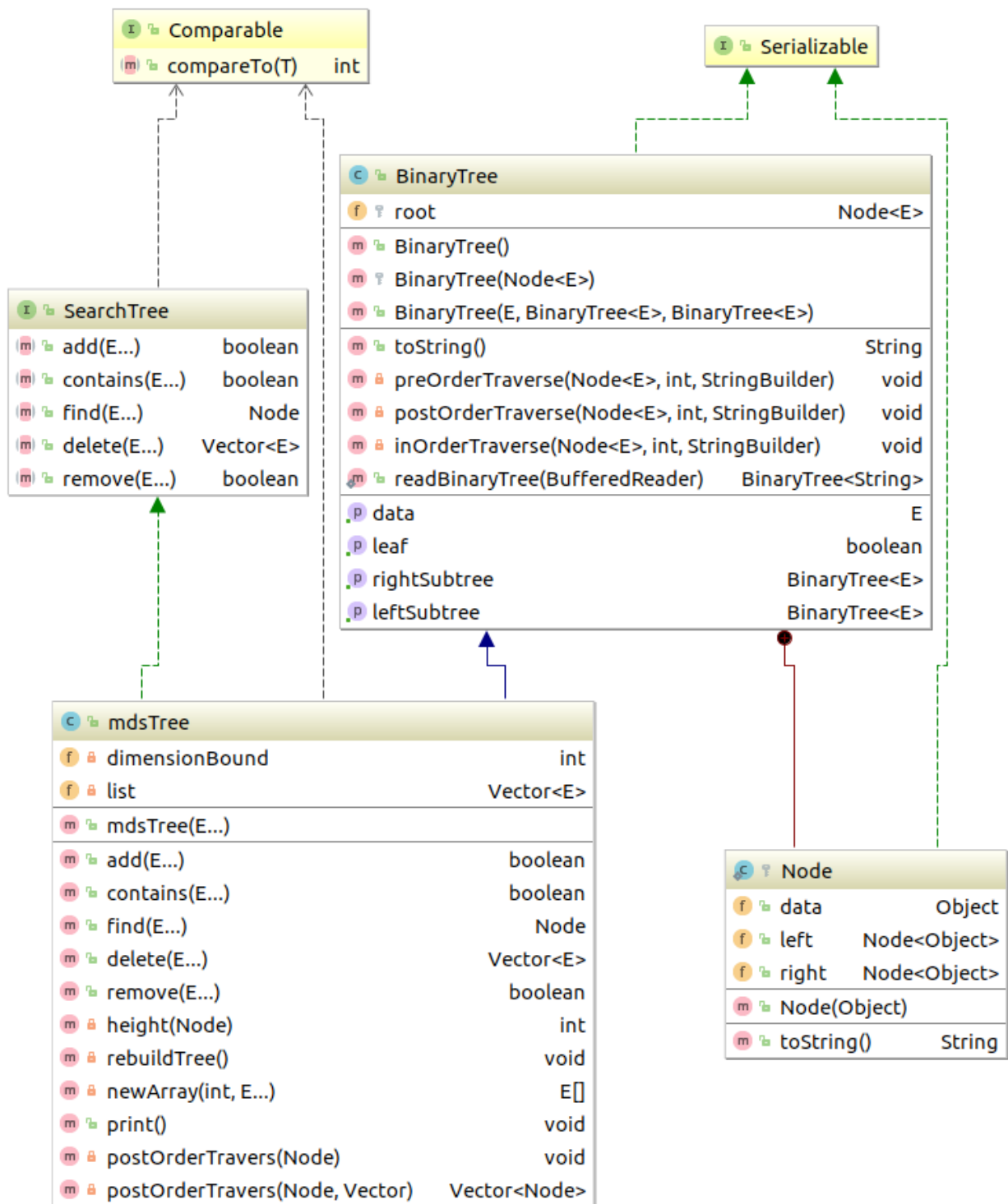
- Birden fazla parametre alabilmek için SearchTree fonksiyonları (E... item) şekline çevrildi.
- Birden fazla eleman olabildiğinden SearchTree deki, delete fonksiyonu Vector return edecek şekilde değiştirildi.
- Yine çoklu parametreden kaynaklı , elemanlar vector şeklinde tutulduğu için Node yapısı E yerine Object alabilecek hale getirildi.
- Karşılaştırmalarda compareTo methodu kullanıldı.

2 METHOD

2.1 Class Diagrams







2.2 Problem Solution Approach

Part 1: “GeneralTree.java” dosyası bu part içindir. BinaryTree’ den extend edilerek yapılmıştır. BinaryTree kitap kodlarından alınmıştır.

add: parent ve child olmak üzere 2 tane String alır. Çocuklar sola, kardeşler sağa olacak şekilde ağaca ekleme yapar. İçinde levelOrderSearch kullanarak parent’ı bulur ve bu parent’a child’ı ekler. levelOrderSearch yerine postOrderSearch te kullanılabilir.

- levelOrderSearch kullanılırsa, Queue kullanarak çalıştığından en kötü durumda tüm elemanlar gezilmiş olur ve bundan dolayı çalışma süresi **O(n)** olur.

- postOrderSearch kullanılırsa eleman bulunsa bile hem sağ ağaç hem sol ağaç gezildiğinden dolayı çalışma süresi **Teta(n)** denebilir.

postOrderSearch: helper method ile çalışması sağlanır. Asıl iş helper’ da olur. Aslında binaryTree ye göre inOrder olarak çalışır fakat bizim aile ağacımıza göre postOrder gezer. Aranılan eleman bulunsa bile tüm ağacı gezer. Gezerken aynı zamanda print eder. Bulunan eleman parametreden gelen Node’da kaydedilir. Tüm ağaç gezildiğinden çalışma süresi **Teta(n)** dir.

levelOrderSearch: Queue yapısı kullanılarak implemant edildi. Root’u queue ya ekleyerek başlar. Queue da eleman kalmayana kadar devam eden döngü içerisinde, queudan çıkarılan elemanın sol tarafı(child) direk, bunun sağ tarafları(kardeşler) null olana kadar queue ya eklenir. Bu sayede aile ağacımızda level olarak gezilmiş olur. Çalışmasını görebilmek için levelOrderPrint aynı işlemin print versiyonudur ve bu amaç için kullanılabilir. İteratif bir döngü olduğundan eleman bulunduğunda döngü kırılır bu sayede erken bitme durumu vardır fakat en kötü durumda çalışma süresi **O(n)** dir.

preOrderTravers: Hem binaryTree için hem de bizim aile ağacı gezilirken aynı sonuçları verir. Yani hani ağaç gezilirse gezilsin sonuç aynı bu yüzden BinaryTree içindeki method kullanıldı. Her halikarda tüm node’lar gezildiğinden çalışma süresi **Teta(n)**.

Part2: “mdsTree.java” dosyası bu part içindir. BinaryTree’yi extend SearchTree yi implement eder. Constructor’ a verilen parametreye göre sonraki elemanların sınırlarını belirler(tree(1,2,3) ise sonrakilerde 3 elemanlı olmalı).Birden fazla parametreyi tutmak için Vector kullanılmıştır. Bu yüzden BinaryTree içindeki Node yapısı Object yapılmıştır. Aynı zamanda birden fazla parametre alabilmek için SearchTree methodlar (E) yerine (E...) alacak şekilde ayarlanmıştır. Aynı zamanda delete methodu parametreler Vector olarak tutulduğundan return değeri Vector olacak şekilde değiştirilmiştir. Herhangi bir düzlem sayısı ve E tipinde çalışabilir.

add: Yollanan elemanlar önceden ağaçta varmı diye contains methodu ile kontrol edilir. Eğer önceden eklenmişse direk false return eder. Eğer tüm elemanlar silindiyse($root == null$) ve add ile ekleme yapılamaya çalışılıyorsa parametre sayısı yeni verilecek iteme göre yeniden belirlenir. Her katmanda kendi düzleminin elamanına bakarak sağa veya sola hareket eder(bu mod işlemi sayesinde sağlanır ($i \% dimensionBound$)). Eğer bakılan düzlemde eşitlik durumu varsa sola gider. Daha sonra null olan yere yeni eleman eklenir. Eleman eklenirse true eklenemezse false return eder. Sürekli sağa veya sola olacak şekilde belli bir tarafa ilerlendiğinden dolayı kabaca **$O(\log n)$** çalışma süresine sahip denilebilir

contains: Veilen elemanın ağaçta ise true değilse false return eder. Add methodundaki gibi kendi düzlemindeki elemana bakarak ağacı gezer. Sürekli sağa veya sola hareket ettiğinden çalışma süresi **$O(\log n)$** dir.

find: Önce contains ile eleman var mı diye kontrol edilir. Varsa önceki methodlardaki gibi hareket ederek Node bulunur ve return edilir. Yine çalışma süresi **$O(\log n)$** dir.

delete: Önce contains ile eleman var mı diye kontrol edilir. Sol ve sağ ağaçlar için simetrik algoritmalar vardır. Sol ağaçtan silerken, eğer silinen elemanın sağında eleman yoksa direk bir sonraki sol referans gösterilir ve silinmiş olur. Eğer sağında eleman varsa bu elemanın en solundaki eleman silinenin yerine koyulur. Bu sayede silinmiş olur. Sağ ağaçtan silme yapılırken soldakinin simetrik(sol yerine sağ , sağ yerine sol) hali kullanılır. Root tan silme yapılırken sağında veya solunda eleman olması, her ikisinde olması veya hiç olmaması durumuna göre farklı şekilde çalışır. Her silme işleminden sonra katmanlar arası denge bozulabileceğinden ağaç rebuildTree fonksiyonu ile yeniden kurulur. RebuildTree fonksiyonundan dolayı çalışma süresi **$O(n)$** dir.

remove: delete fonksiyonunu çağırır eğer null'sa false değilse true return eder. Delete methodundan dolayı **$O(n)$** dir.

rebuildTree: Silme işleminden sonra ağacı postOrder gezer ve yeniden kurar. PostOrder 'dan dolayı **$O(n)$** çalışır.

3 RESULT

3.1 Test Cases

Main test için ekstra girdi olmadığı için test dosyası yazılmadı. Main'de basit senaryolar var.

Diğer class lar için kendi isimlerindeki test dosyaları, unit test için hazırlandı.

3.2 Running Results

```
Run Main
/usr/lib/jvm/java-8-oracle/bin/java ...
parent 1 2 3 11 12 13 21 22 31 111 221 311 312


111 11 12 13 1 21 221 22 2 311 312 31 3 parent

|
parent
  1
    11
      111
        null
        null
      12
        null
        13
          null
          null
    2
      21
        null
        22
          221
            null
            null
            null
      3
        31
          311
            null
            312
              null
              null
            null
          null
        null
      null
```


-----part2-----

```
8 90 900
7 8 9
13 6 7
2 3 4
61 61 61
78 13 19
16 17 18
15 12 11
13 15 71
node : [7, 8, 9]
node : [16, 17, 18]
node : [61, 61, 61]
2 3 4
7 8 9
15 12 11
16 17 18
78 13 19
61 61 61
13 6 7
8 90 900
```

Process finished with exit code 0

 Compilation completed successfully in 865ms (15 minutes ago)