Gebze Technical University
Computer Engineering

CSE 222 - 2018 Spring

HOMEWORK 4 REPORT

HALİL ONUR ÇEÇEN
161044057

Course Assistant: Mehmet Burak KOCA

1 INTRODUCTION

1.1 Problem Definition

1.1.1 PART 1

I need to construct a General Tree structure using Binary Tree representation. Which means a node can only have 2 children. And this structure should have include ordinary general tree methods. To achieve this goal I have to extend regular Binary Tree class which can be found on course book.

1.1.2 PART 2

In this part, I need to construct a K Dimensional Tree using both binary tree and search tree class and interface respectfully. Also its methods must be implemented according to search tree methodology.

1.2 System Requirements

1.2.1 PART 1

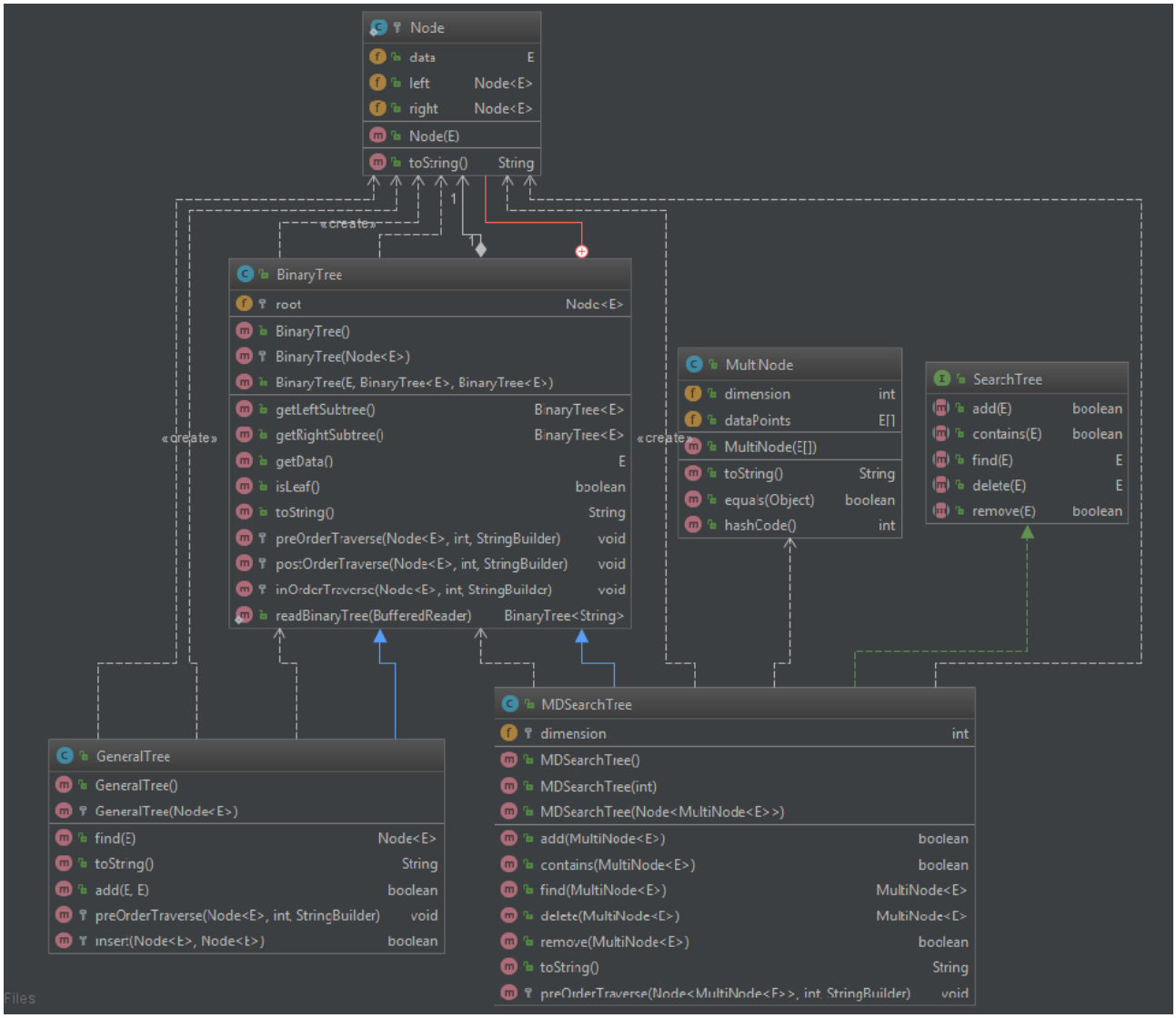
General Tree has 2 constructors to initialize itself. First one takes no parameters and creates an empty general tree. To add first item to the tree, you must give null as parent node else it won't work. And second constructor takes a Node parameter and makes it root of itself. But since Node type is protected, user can't reach Node type to use it. So it's mainly for use of methods itself.
Also, since Node can't be reached from outside, add method takes value instead of node.

1.2.2 PART 2

MDSearchTree (Multi-Dimensional Search Tree) has 3 constructors as requested in assignment (Default, dimension, root Node).
This class uses a MultiNode class which I implemented to include multiple values in a single Node. So, while sending values to methods that take value, you have to send an instance of MultiNode<E> class with its constructor that takes an E array (example of sending instances can be found in Main test).

2 METHOD

2.1 Class Diagrams



Both General Tree and MDSearchTree uses Node from BinaryTree. But MDSearchTree also uses MultiNode.

2.2 Use Case Diagrams

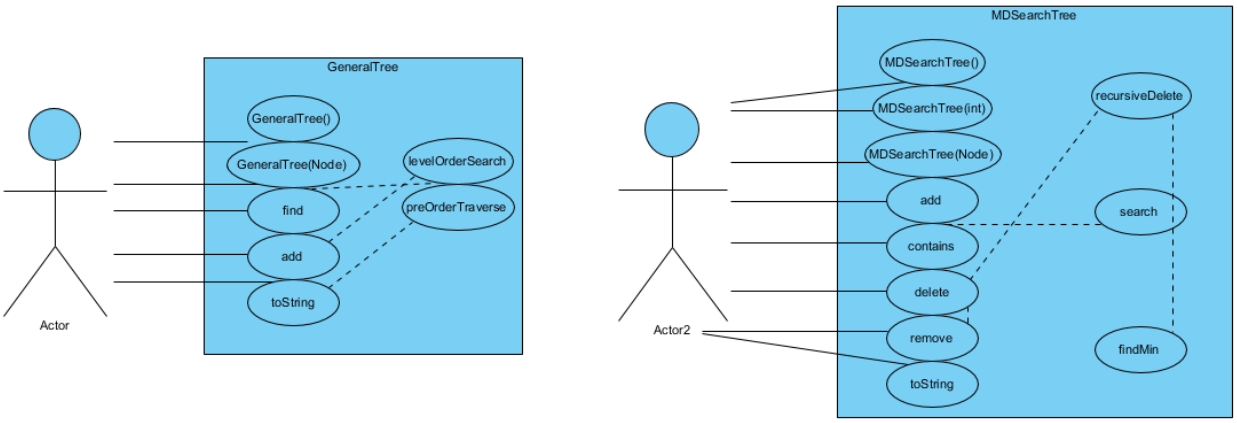


Diagram shows interactable user methods and their relations with helper methods.

2.3 Other Diagrams

All diagrams can be found in Diagrams and Screenshot folder.

2.4 Problem Solution Approach

2.4.1 PART 1

For the sake of Tree methodology, most of my methods except add implemented recursively. Add method is a kind of adapter for insert method to construct Node types of given data. And insert method finds given data and goes to the last sibling of child iteratively. Most of expressions are self-explanatory or has their description in Javadoc.

2.4.2 PART 2

To have different types of data in MDSearchTree and have multiple of them in one Node, I created an adapter class between Node and given E type as MultiNode<E>. While constructing MDSearchTree, its generic type E (which is extending Comparable<E>) is used while extending BinarySearch and SearchTree as <MultiNode<E> >.
Also, while operating through tree I have kept an index for the dimension of current level. This provides use of index operator while comparing according to the level.
Most of my methods are recursive since it's easier to think and code while going over trees.
Note: I have added another method called findMin to help deletion process. It finds Minimum value of desired dimension to replace deleted item between Nodes.

3 RESULT

3.1 Test Cases

Main method test case:

3.1.1 PART 1

- Constructs Integer type General Tree and adds 9 items.
- Tests add methods return value
- Tests levelOrderSearch through find method
- Tests preorderTraverse through toString method
- Constructs String type General Tree and adds 6 items.

3.1.2 PART 2

- Constructs Integer type default MDSearchTree.
- Tests add method by adding 6 3-dimensional value.
- Tests contains and delete method by checking its validity and deleting it.
- Prints Tree by preOrderTraverse.
- Note: Other Constructors will be tested through Unit Test.

Note: Unit tests of every part can be found in test folder and Javadoc.

3.2 Running Results

Part 1

```
PART 1 Tests:
Successful parent insert.
Can't add 7 to parent 6 because it already exists.
Node: 15
5
  6
    4
    1
    7
  8
  10
  15
    14
      13

Onur
  Ahmet
    Aslı
  Dilara
  Yasir
    Fatih
```

Part 2

```
PART 2 Tests:
Before delete
[18, 27, 44]
  [13, 30, 32]
    [5, 10, 15]
      [8, 4, 6]
    [1, 31, 3]
  [21, 37, 45]

After delete
[18, 27, 44]
  [13, 30, 32]
    [5, 10, 15]
      [8, 4, 6]
  [21, 37, 45]
```

NOTE: All images, diagrams can be found under Screenshots and Diagrams folder.

3.3 Time Complexity

Suppose n = number of courses;

3.3.1 PART 1

Constructor = O(1)

levelOrderSearch= T(n) = T(n-1) + 4 (n = level count) (4 = if evaluation and variable assignment)

postOrderSearch= Same as levelOrder
getByRange(k) = T(k) = O(k)

3.3.2 PART 2

Constructor = T(n) = O(1)
Insert = T(n) = O(levelOrderSearch(n) + k) (k = Sibling count)
Find = T(n) = O(logn)
Contains = t(n) = O(logn)
Delete = T(n) = O(logn x O(findMin))
Findmin = T(n) = Couldn’t calculate