

The purpose of this assignment is to practice your knowledge of binary search trees. The assignment must be completed in pairs. As always, do not modify any package names, method names, or file names. Read the analysis document before starting.

1. The Problem

We have been asked to construct a program the spell-checks a document, given a "dictionary" of valid words.

We know that the key to an efficient spell-checker is an efficient mechanism for storing and searching the dictionary. We decide to maintain the dictionary (simply a list of strings) in sorted order, so that searching the list will be efficient.

Furthermore, we decide to store the dictionary in a binary search tree (BST) because all operations on the BST are *potentially* very fast.

We must first implement a BST of generically-typed items. The implementation should include a class to represent a binary tree node and a class to represent a BST (a collection of binary tree nodes). ***Please note that there is no requirement that the BST be balanced.***

We must then apply the BST to the problem of spell-checking a document, using a BST to store a dictionary of strings and looking up each document string in the BST. The implementation should include a class to represent a spell-checker and a class to demonstrate the spell-checker for various dictionary and document files.

2. Requirements

1. SortedSetinterface

The SortedSet interface has been provided for you. It contains all of the operations that your `BinarySearchTree` class must implement.

2. BinarySearchTreeclass

To the `assignment8` package add a `BinarySearchTree` class that implements the `SortedSet` interface. The class must represent a binary search tree. (Construct a class to represent a binary node, as needed.)

To get you started, the header for your `BinarySearchTree` class is given here:

NAVIGATION

Home

■ My home

Site pages

My profile

My courses

Computer Science

Previous Semester

CS 1410-1-S13

CS_2100_S_13

CS2420-S13

Participants

General

Getting started;
Java review

Generic
programming;
Object Oriented
Programming

Algorithm
analysis; Data
Structures

Basic Sorting
Algorithms

Recursive Sorting
Algorithms

Linked Lists

Stacks and
Queues

```
public class BinarySearchTree<Type extends Comparable<? super  
Type>> implements SortedSet<Type>
```

Note that the type of data stored in your BST must be comparable (meaning it has a `.compareTo()` method), and you don't need a Comparator.

Add a no-parameter constructor `BinarySearchTree()`, which creates an empty BST.

3. SpellCheckerClass

The SpellChecker class has been started for you. Fill in the methods as directed. Do not modify the signatures of the existing methods.

4. SpellCheckerDemoClass

The SpellCheckerDemo class has been provided for you. It should be used as a starting point for running your SpellChecker implementation; however, this class is not a tester. You should perform exhaustive testing in a separate tester class.

SpellCheckerDemo references several files.

- A small dictionary file, dictionary.txt
- A document file, hello world.txt
- A document file, good luck.txt

It is **strongly** recommended that you add a method to your `BinarySearchTree` class `writeDot(String filename)`, which generates a dot file with the given name from your BST. See the code example from lecture 15 for a complete implementation. You can mostly copy/paste, but you may have to change a few variable names to match your own node class. For more information and examples on dot, see this page.

Create your own tests and submit them with your program.

3. When preliminary coding is complete and your program compiles without error or warning, test the program thoroughly and systematically.

Your code should be well-commented (Javadoc comments are recommended) and formatted such that it is clear and easy to read. Be sure to put the names of both programming partners in the header comment of each file.

Trees

 Lab 5

 Lab Files

 debugperspecti

 Slides


 Tree Traversal
Applet


 Code Demo

 Slides

 Code Demo

 **Assignment 8**

 Analysis
Document

 Using the DOT
tool

 example.dot

 Assignment 8
files

Graphs

Spring Break!

Hash Tables

Binary Heaps

File Compression

Comprehensive
Project;

Multithreading

Wrap Up

Final Exam and
Review

29 April - 5 May

Zip your source code files (.java files only) and **upload the zip file here by 5pm on March 7**. Please submit just one solution per pair.

4. [Analysis Document](#) (must be written and submitted by each programming partner) ***due March 7 at 5pm***

Due date: Friday, 8 March 2013, 11:55 PM

Submission feedback



Poonam Ekhelikar

Wednesday, 20 March 2013, 11:17 PM

Grade: 95.00 / 100.00

--Scoring--

Successful BST: 38/40

Successful Spell Checker: 25/25

Quality of student tests and overall style: 10/10

Total: 73/75

Total: 95/100

TA Comments:



Assignment8Analysis.pdf

Submission

Assignment administration

Submission

Course administration

My profile settings

No further submissions are allowed.