Assignment 9

Reminder: Your code is to be designed and written by only you and not to be shared with anyone else. See the Course Outline for details explaining the policies on Academic Integrity. Submissions that violate the Academic Integrity policy will be forwarded directly to the Computer Science Academic Integrity Committee.
All materials provided to you for this work are copyrighted, these and all solutions you create for this work cannot be shared in any form (digital, printed or otherwise). Any violations of this will be investigated and reported to Academic Integrity.

Objectives

- Introduction to the Binary Search Tree ADT
- More practice with generics
- Introduction to tree traversals
- Introduction to the Map ADT.

Introduction

In this assignment you will implement a binary search tree for which each node will contain both a key and a value associated with that key. The elements in the tree will be ordered by the key stored at each node. After you have completed implementing the binary search tree, we will use that implementation to implement the Map ADT.

Submission and Grading

Submit your `BinarySearchTree.java` and `BSTMap.java` and with the completed methods through the assignment link in BrightSpace.

- You **must** name your methods as specified in the given interface and as used in `A9Tester.java` or you will receive a **zero grade** as the tester will not compile.
- If you chose not to complete some of the methods required, you **must at least provide a stub for the incomplete method** in order for our tester to compile.
- If you submit files that do not compile with our tester (ie. an incorrect filename, missing method, etc) you will receive a **zero grade** for the assignment.
- Your code must **not** be written to specifically pass the test cases in the testers, instead, it must work on other inputs. We may change the input values when we run the tests and we will inspect your code for hard-coded solutions.
- **ALL late** and **incorrect** submissions will be given a **ZERO** grade.

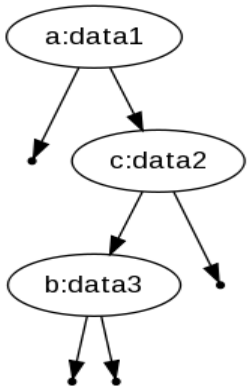
Getting Started

1. Read the comments in `BinarySearchTree.java` carefully and complete the method implementations until the tree tests pass in `A9Tester.java`
   Debugging tips are provided in Part I below.
2. Uncomment the map test method calls in the `main` of `A9Tester.java`
3. Read `Map.java` carefully and complete the method implementations in `BSTMap.java` until all the map tests pass in `A9Tester.java`
   Implementation tips are provided in Part II below.

Part I - BinarySearchTree

Your binary search tree implementation will use generics to allow users of the tree to specify the type for both the key and the value. For example, the following code produces the tree shown:

```
BinarySearchTree<String, String>   t1;
t1 = new BinarySearchTree<String,String>();
t1.insert("a", "data1");
t1.insert("c", "data2");
t1.insert("b", "data3");
```



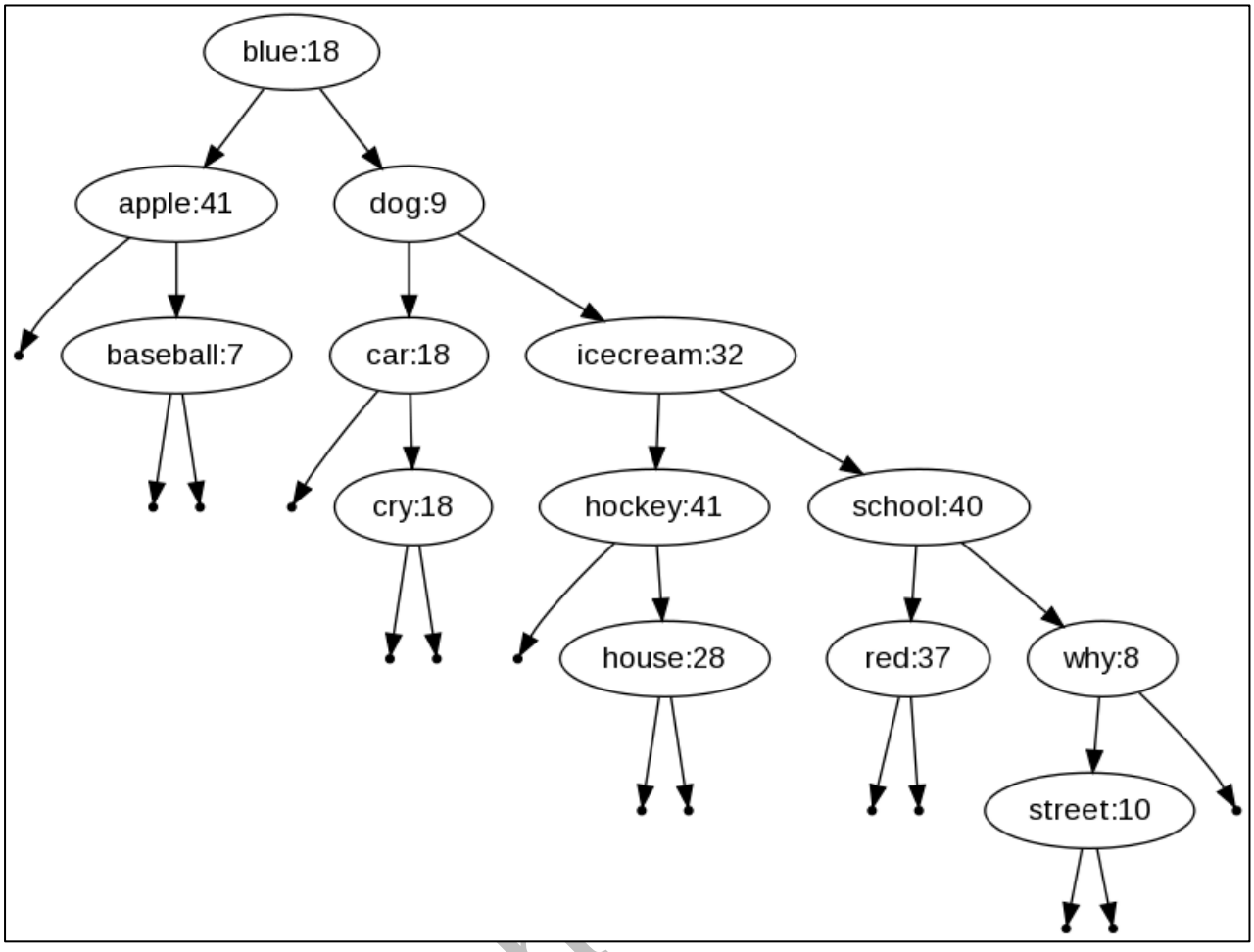
When testing your code, you may find it useful to look at the trees created and used by `A9Tester.java` (A visual representation of each of the four test trees is provided on the following pages of this assignment description.)
Since it is helpful to be able to visualize trees, you've been provided with a `TreeView` class which takes an instance of a binary search tree as the constructor's parameter. For example, if after the code shown above you use the following code:

```
TreeView<String,String> v1 = new TreeView(t1);
v1.dotPrint();
```

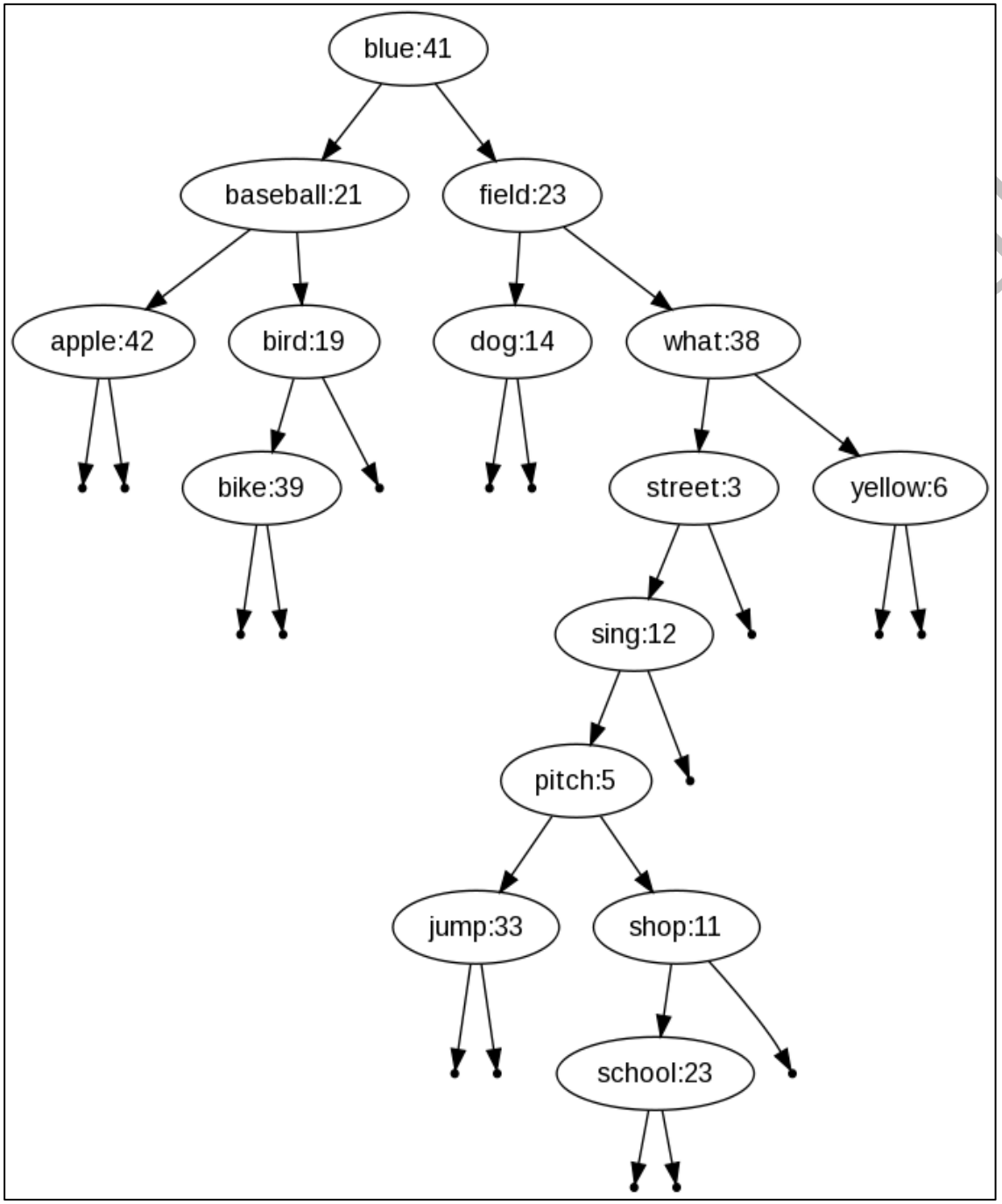
The result will be the contents of a dot file printed to the standard output. You can copy and paste that output text into the following web page to get a picture of your tree:

http://www.webgraphviz.com/

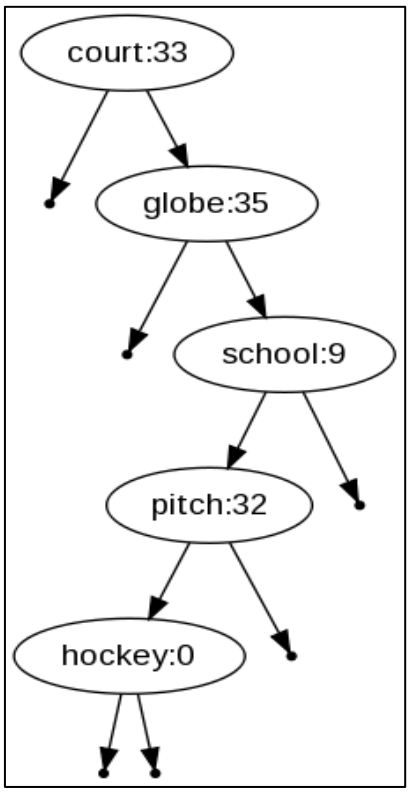
The following images are visual representations of the 4 trees generated using the `TreeView` class as described above in `A9Tester.java`



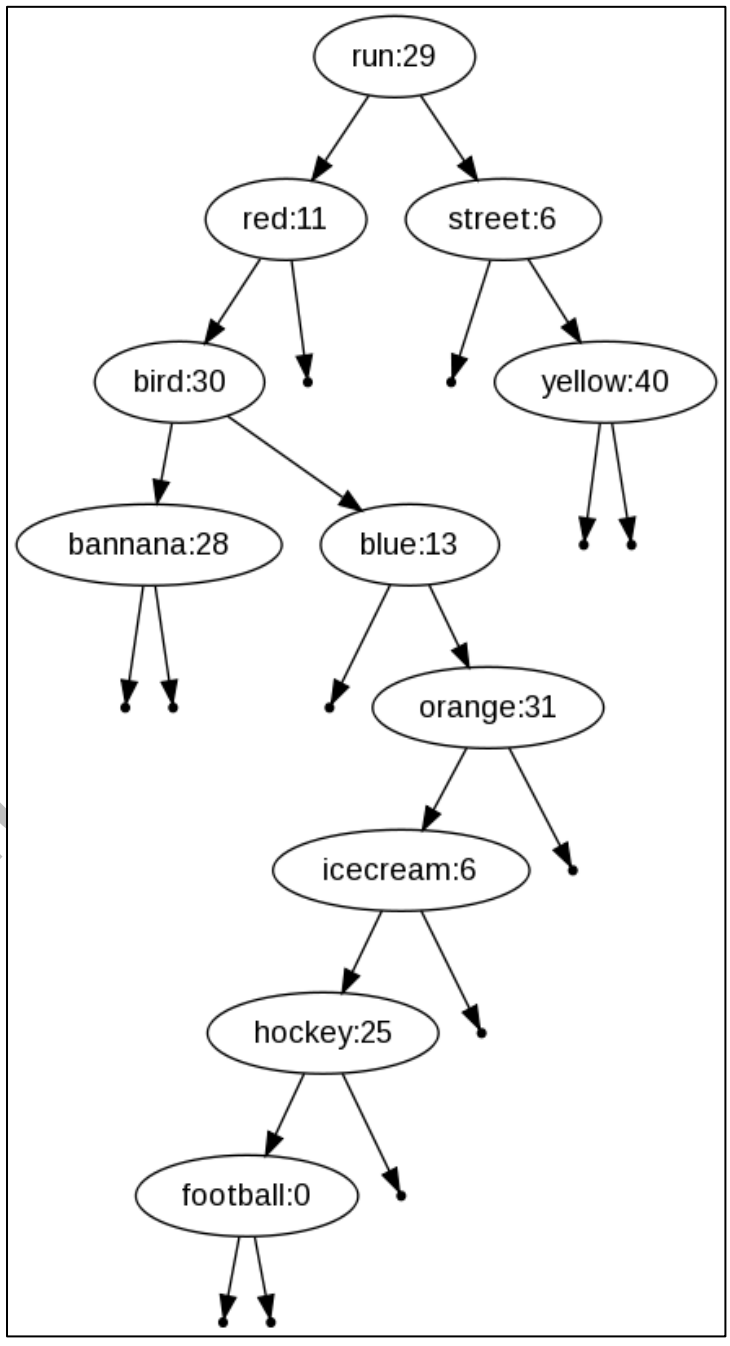
Tree 1


blue:41
baseball:21
field:23
apple:42
bird:19
dog:14
what:38
bike:39
street:3
yellow:6
sing:12
pitch:5
jump:33
shop:11
school:23

Tree 2



Tree3



Tree 4

Part II – BSTMap

Once you've finished implementing the Binary Search Tree, you will use this to implement the Map ADT in the file `BSTMap.java`. The Map interface is specified in `Map.java`, and provides two main operations:

- `get(key)` – return the value stored at key
- `put(key,value)` – insert the key/value pair into the map. If the key already exists, update the value stored at key to be the new value.

You have to write very little code to accomplish this – simply create an instance of your `BinarySearchTree` in the `BSTMap` constructor and then use methods you have already completed in the `BinarySearchTree` to provide the services required by the Map[1].

(In your instructor's sample solution, all methods except `containsKey` are only one line. `containsKey` contains seven lines.)

[1]This is a common practice. You can read about it here: https://en.wikipedia.org/wiki/Adapter_pattern