CSE 12 HW9 --- Binary Search Tree + Data Persistence

Disclaimer

Most of the code if not all that is presented in this note should be interoperated as pseudocode!

Key Idea:

We would like to store the tree so that when the program terminate, we would still have the data that we inserted in the tree.

Writing and Reading Files

In this homework, you are going to use *RandomAccessFile* library.

Here is the documentation of this library https://docs.oracle.com/javase/7/docs/api/java/io/RandomAccessFile.html.

I also found these two tutorials quite useful:

- https://www.journaldev.com/921/java-randomaccessfile-example.
- https://www.tutorialspoint.com/java/io/java_io_randomaccessfile.htm

Thoughts

Think the file as an array.

- You would need to know where to access the file (index in an array), and you would need to know how long you would need to read from the file (type of the element stored in the array).
- The only difference is you would need to *seek* to the position that you want to start reading (analogous to accessing the index of the array, but you need to write one line of code to access).
- The snippet of code you would need is `fio.seek(position);`

TODO:

Driver.java

Add the methods that you would need based on the `Variable` class of the `Calculator.java`. In particular:

UCSDStudent Class

- copy:
  - Create a copy of the current UCSDStudent by creating a new object holds the same data
- jettison:
  - Jettison the tracker and set it to `null`

- getName:
    - Return the name
- getTrimName
    - Return the trimmed name
- equals
    - Check whether the two students have the same name
- isLessThan
    - Check whether one is greater than another in alphabetical order
- read
    - Read a UCSDStudent from the disk (refer to variable)
- write
    - Write a UCSDStudent to the disk (refer to variable)

Main Method

- You would need to implement the 'f' case for inputing from ASCII file.
    - You should prompt the user `"Please enter file name for commands: "`
        - Look at other cases to see how we do prompts
    - You would need to read in the file name
        - `MyLib.getline (is)` to read
    - You would need to create a new input in stream by calling `new FileReader(filename);`
    - You would need to redirect the output to `/dev/null` by creating a `new FlushingFileWriter (filename);`
    - You would need set the `readingFrom` variable to be `FILE`.
- You would need to handle what to do if you read an EOF
    - If you are reading from the key board, you should break the `while` loop to terminate the program.
    - If you are reading from a file, you would need to change the input stream back to `System.in` and change the out stream back to be `System.out`
        - How to do them? Take a look at the lines before the `while loop`

Tree.java

General Note

Because how the Java library is designed, you would always need to use the following try catch block whenever you would like to read/write from/to the disk

```
try {
  // fio functions
}
catch (IOException ioe) {
  System.err.println("IOException in <Method Name>");
}
```

Datafile

0000000 0000 0000 0000 5000 0000 0000 0000 0000

0000020 0e00 2061 2020 2020 2020 2020 2020 2020

0000040 0000 0000 0000 0100 0000 0000 0000 0000

0000060 0000 0000 0000 0000 0000 0000 0000 0000

0000100 0000 0000 0000 0000 0000 0000 0000 1000

0000120

Tree Method

- Tree Constructor
    - You first want to assign the sample to the this.sample.
    - Then you would need create a sampleNode by calling the default constructor of the TNode (we have given to you)
    - Then you need read the file to get the tree in.
    - Create a `RandomAccessFile` by the datafile parameter that you passed in, set the mode to be read and write. AKA this:

      ```
      fio = new RandomAccessFile (datafile, "rw");
      ```

    - Then two cases:
        - Datafile file is empty (how to check the file is empty?)
            - If the files is empty, you write the root and occupancy to the beginning of the disk.

              ```
              fio.seek(BEGIN); // Go the the begining of the file (BEGIN = 0)
              fio.writeLong (root); // Write a long to the disk, the long is the root
              fio.writeLong (occupancy);
              ```

            - Set the root position to be the end of the file (because that will be where your first node locate)

```
root = fio.getFilePointer(); // Get the current position of the file
```

    - Set the occupancy to 0
  - Data file is not empty
    - Go to the beginning of the file,

```
fio.seek(BEGIN); // Go the the begining of the file (BEGIN = 0)
```

    - Read in the root and the occupancy.

```
fio.readLong(); // Read a long from the disk starting at the position you seeked
```

- isEmpty:
- insert:
  - Call `incrementOperation()`
  - Special case: Empty tree:
    - Create a new Node by calling the TNode write ctor
    - `jettionTNode` after you write the TNode.
  - Not Empty:
    - Create a root box
    - Call `searchTree` on the sampleNode passing in the element and the rootBox and set the mode to be `"insert"`.
    - Update the `root`
- lookup:
  - Call `incrementOperation()`.
  - If you implement it recursively, should be similar to remove.
  - If you implement it in a loop way, translate your hw7 to here.
- remove:
  - Call `incrementOperation()`
  - Check whether the tree is empty, if so just `return null`.
  - Else, create a rootBox, call `searchTree` on `sampleNode` passing in the rootBox, element, and sent the mode to be `"remove"`.
  - Update root's position.
  - *If the tree is empty after remove, you would need to call resetRoot to update the root position so that you are writing at the end of the file when you insert new data.*
  - Return result.
- resetRoot:
  - Seek to the end of the file.

```
root = fio.seek (fio.length ());
```

   - Set your root to be the position of the end of the file.
   - Why?
     - Empty Tree, then the root will start at the end of the file when insert a new element.
- write:
  - Seek to he beginning of the file
  - Write the root
  - Write the occupancy

TNode Method

- read
  - Because you are preforming a read, you need to call `incrementCost ();`
  - This function is called from TNode read ctor
  - Seek to the position that is passed in.
  - Call `data.read(fio);` to read in the `element`.
  - Then read in each field of the TNode by calling `readLong ()`
- write
  - Because you are writing to the file, you need to call `incrementCost ()`;
  - Seek to the `position` (this time the `position` is the data field belongs to the TNode)
  - Call `data.write(fio);` to invoke the write method of the element to write it to the disk.
  - Then just write each field to the disk by calling `writeLong`
- TNode read Ctor
  - Make a copy of the data of the sample by calling `data = (Whatever) sample.copy();` to generate a new data for the TNode to use.
  - Delegate the rest work to the read method
- TNode write Ctor
  - Set the `data` to be `element`.
  - Init the `left` and `right` to be 0 (which indicate there is no left and right child of the TNode)
  - increase the occupancy
  - Set the `position` to be the current end of the file.
  - Delegate to the write method.
- lookup
  - Implementation depends on whether you do it loop based or recursively.
  - Personally I recommend you do it recursively.
- insert
  - The return type is `Whatever`, so you just return the `element` back if the insertion is succeed.
  - Figure out where you want to go down the tree.

- If it is duplicate insertion, after you resign the data to be the element, make sure you call `write()` so that the TNode is written to the disk.
- If you have a left/right to go, create the box and call `searchTree` on this passing in the box.
  - The position that is stored in the box will let `searchTree` know where to read a Node in.
  - Don't forget to update your position after `searchTree` returns.
- If you don't have a left/right to go, create a TNode by calling the `write` ctor. Update parent's left/right by the newly created TNode position.
  - Think: When do you know you don't have a left/right to go?

- remove
  - You should return `null` if removal failed and return **a copy** of the `data` if you found the one to remove.
  - The general idea is similar to `insert`.
  - Review your hw8, follow the dev guide and change the calls to `searchTree` with the correct mode being passed in.
  - A note to `RARM`
    - Instead of passing in a targetTNode, this time `searchTree` with RARM mode will actually return the data in the successor. So you should assign `data` to be the result of `searchTree`.
    - The `setHeightAndBalance` method will call write, but if you are called `fromSHB`, *you would need to call `write` explicitly so that the successor's data got written to the disk.*
    - Of course, you would need to first store the `result` that you want to return before reassign the `data`.
- replaceAndRemoveMin
  - Same idea as hw8, just return the data when found, rather than set it to `targertTNode`.
- setHeightAndBalance
  - If you have a left or have a right, read the child in by calling the TNode read ctor with the right and left position.
    - Check out `seachTree`
  - Once extracted the `height` information of your child, `jettisonTNode` because it will no longer be used.
  - Do the calculation
  - If the `THRESHOLD` is exceeded, call `remove` function and then call `searchTree` passing in the COPY of the element that you just removed as parameter.
    - The `copy` function of the `element` will return a Base/Whatever, so you need to cast it to a `Whatever` upon insertion.
  - Write the TNode before the end of SHAB

Final Note

Start early! Good luck!