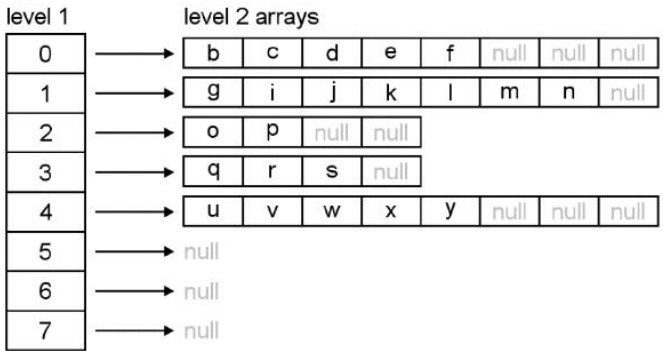
CSCI-290 *Project Part 3*

Ragged Array List - Find

Objectives: Building a new data structure (the find methods). In the next 3 programming assignments you will be creating a new data structure called a "Ragged Array List". A diagram of a small example is shown on the right. There is a level 1 array that contains references to a series of level 2 arrays. Together these contain the data elements kept in sorted order. This example contains 22 strings.



It is called "ragged" because the second level arrays do not all need to be the same size. It has strong performance advantages over linked lists and array lists when the data must be kept ordered. As with array lists, the arrays grow as more data is added to them. The internal implementation is completely hidden from the external user. To an external user it is just a collection of objects kept in sorted order.

It is very important that you only implement the methods that you are told to implement in each part of this series of assignments. Do not get ahead of the assigned tasks!

A Few Data Structure Details:

1. This is an ordered list. Items are kept in sorted order based on a comparator.
2. Duplicates are allowed. When searching for a match, the search should always return the earliest matching item. If adding a duplicate item, it should always be added after the last matching item. (E.g. if you were to add another 'b' to the above diagram, it would be inserted between the 'b' and 'c'.)
3. You will be creating a generic data structure that hold objects of a designated type, eventually Songs, but Strings for now.
4. You will be using a Comparator to compare items.
5. An invariant of the design is that there is always space available to insert another item. Note above that each of the level 2 arrays has at least one unused slot, and also that the level 1 array has at least one unused slot. This invariant will make the code a little easier to write.

Warning: You must use the data structure that is part of the assignment to accomplish the tasks. Use of Arrays or ArrayLists where they are not indicated will result in 0 points

Setting up the starting and testing code:

1. Starting files are on Brightspace:
   - RaggedArrayList.java
   - RALtester.jar
   - Scaffold.jar
2. Create a new project called Part03 inside your CSCI290Projects folder. Then Copy the student folder from the Part2/src folder and Paste it inside the Part03/src folder. Save the 3 new files to the student folder as well.
3. Choose File - Project Properties - click Libraries in the categories pane and click on the Run tab. Click [Add Jar/Folder] and add the RALtester.jar to the project. Then add Scaffold.jar.
4. The RALtester.jar and Scaffold.jar contain code that will test your findStart() and findEnd() methods that you will be writing for this assignment. Usually it is hard to build a new data structure from scratch because you need to get several parts working before you can start testing anything. In the Part 4 you are going to be writing the add() method, which will use findEnd() to find the insertion position. The RALtester,jar will allow you to test and fully debug your find methods without actually having any code yet to build a ragged array list. It works by injecting a pre-built ragged array list into your ragged array list class and then calling your find methods to test them.
5. To use it for testing go to the run configuration on the Project Properties tab and set the Main Class to RALtester.RALtester (check the spelling or else it won't find the main class)
6. Run (The starting code is working code, although it will not find correct answers.)
7. By default the tester will run all test cases. If you want to run just a specific test case for debugging, you can put it in the Arguments textbox of the run configuration. E.g.: `test1.txt` If you look in the RALtester.jar, you will see a directory called tests and inside that are the test files for your project. There are 28 test files.

Understand the starting code RaggedArrayList.java

1. Spend some time looking at the starting code.
   The RaggedArrayList has fields:

int size	the number of objects stored
Object[ ] l1Array	really is an L2Array[ ] (declared this way due to Java limitations)
int l1NumUsed	the number of slots used so far in l1Array
Comparator<E>comp	the comparator used for comparing items

   Nested classes:

class L2Array	used to store the objects in the level 2 array
class ListLoc	used internally to store a pair of level 1 and level 2 indices
class Itr	used internally to implement an iterator

   Methods:

RaggedArrayList(Comparator<E>c)	constructor that accepts a Comparator
size()	returns the current number of item held
clear()	resets it to empty
findFront(E item)	used internally to find the first match
findEnd(E item)	used internally to find the position after the last match
add(E item)	insert a new item
contains(E item)	search for match
toArray(E[ ] a)	copy into a basic array
subList(E from, E to)	return a subset as a new ragged array list
iterator()	return an iterator

   The L2Array has:

E[ ] items	an array of items for this row
int numUsed	the number of items stored so far in this row
L2Array(int capacity)	constructor that accepts the capacity for this row

   The ListLoc has:

int level1Index, level2Index	the coordinates of a location in the ragged array list
ListLoc(int l1, int l2)	a constructor specifying a location
equals(Object otherListLoc)	compare two ListLoc's
moveToNext()	move to the next item in the ragged array list

The Iterator has:

ListLoc loc	the current position of the iterator
Itr()	constructor that starts at the first item
hasNext()	check if another item exists
next()	return the next item
remove()	not actually implemented, but needed to satisfy "implement"

Fields, internal methods, and nested classes should be private, however they are declared public to that the RALtester.jar can test them. Many parts of this code are already completed and marked as (done). Don't modify those parts!

Task 1: write findFront()

1. Be sure to appropriately document everything you write!!
2. Implement the findFront(E item) method. It should search for the specified item and return its location as a ListLoc.
3. If there are one or more matches, it should return the earliest match in the list. In the example above:
   findFront(k) should return ListLoc(1,3)
   findFront(n) should return ListLoc(1,6)
4. If there are no matches it should return where that item could be inserted. In the example above:
   findFront(a) should return ListLoc(0,0)
   findFront(h) should return ListLoc(1,1)
   findFront(t) should return either ListLoc(3,3) or ListLoc(4,0), both are correct insertion points
   findFront(z) should return ListLoc(4,5)
5. A performance requirement is to use at most 1 comparison in any level 2 array until you find the level 2 array containing the target. Start with linear search and get that working. For extra credit you can use binary search to make it even faster.
6. Coding Note: Because the L1Array is an array of Objects, you will need a cast when using its contents. e.g. `L2Array l2Array = (L2Array)l1Array[i];`
7. Keep it simple. My code was only 14 lines (not counting comments, blanks, and ending }'s).
8. Fully test and debug this with the RALtester.jar before continuing.

Task 2: write findEnd()

1. Implement the findEnd(E item) method. It should search for the next location after any matching items. In the example above:
   findFront(k) should return ListLoc(1,4)
   findFront(n) should return either ListLoc(1,7) or ListLoc(2,0)

2. If there are no matches it should return where that item could be inserted. In the example above:
   findFront(a) should return ListLoc(0,0)
   findFront(h) should return ListLoc(1,1)
   findFront(t) should return either ListLoc(3,3) or ListLoc(4,0), both are correct insertion points
   findFront(z) should return ListLoc(4,5)
3. A performance requirement is again to use at most 1 comparison in any level 2 array until you find the level 2 array containing the target.
4. Keep it simple. Your code will be very similar to your findFront().
5. Fully test and debug this with the RALtester.jar.

What to turn in:

Written part:

Note: you may have to install pdfCreator to do this, but it's well worth the effort. It adds a "printer" to your computer that prints to pdf.

1. Be sure that any new code is attributed to an individual or the team.
2. Click on the Report tab in the RALtester it should show the results for all tests. Choose print on the tester report. Select your pdf printer. Click [Properties]. On the Properties dialog click the Layout tab. For this one leave the orientation set to Portrait. Click the [Advanced] button. Everything that is underlined can be changed. Change Scaling under Graphic and change 100% to about 60%. Click [OK] [OK] and [Print]. Name the pdf file RALtesterFull.pdf and save it to the student package directory.
3. Click on the test12.txt tab. Make sure the radio button next to Result is chosen and the radio button next to Picture is chosen. Print it Landscaped and scaled to about 60% as described above. Save the pdf as RALtest12.pdf.
4. Thinking about your code, what is the Big-O order of your find methods if the ragged array list happens to have a level 1 array with $\sqrt{N}$ rows used, and each level2 array contains $\sqrt{N}$ items? Justify your answer!
5. Any incomplete parts, or any known bugs in your code should be documented and reported in your write up.

Electronic submission:

- I will create groups and put you into them. Anyone in the group can upload files.
- Print your write up to pdf name it Project3.pdf be sure that the names of the team members are in the write up at the top of the first page.
- Upload Project3.pdf, RALtesterFull.pdf, RALtester12.pdf, and RaggedArrayList.java files to Brightspace without zipping them. The pdfs can be combined into one pdf if you have that capability.

Grading:

Written part	10 points
findFront()	60 points
findEnd()	30 points

I expect your programs to be properly commented and use good style. Points may be deducted for egregious disregard of these matters. Every method in every class must have complete documentation including authorship. A full an complete revision history should be at the top of every class file!

Extra Credit (50 points): Do not try this until you have the regular assignment completely working. Make a copy of the project to work on for when you realize this is a LOT harder than you had expected.

- Instead of sequential searching, use binary search on both the level 1 and level 2 arrays.
- Explain the new Big-O time for your find methods.
- Will your code still work if there are duplicates?
- Will your Big-O execution time change if there are lots of duplicates?