CS 345 (Fall 18): Computer Organization

Project #2

Heaps and Heapsort

due at 5pm, Thu 20 Sep 2018

1 Introduction

In this project, you'll be implementing a max heap - and also practicing with `dot`, a tool which allows you to easily generate pictures of graphs (that is, the things with nodes and edges, such as trees). It will be very useful for debugging your heap!

1.1 What You Will Implement

You will implement two classes: `Proj02_HeapSort`, which implements `Proj01_Sort`, and `Proj02_MaxHeap`, which represents a heap but not necessarily a sort algorithm.

You are **required** to have these two classes use common code: in particular, you are only allowed to implement one version of the bubble-up code, along with one version of the bubble-down code. Exactly **how** you share code is up to you; one option is to have Heapsort create a Heap - but other options are possible.

1.2 Test Structure

Your code will be tested in three ways. First, I have provided `Proj02_SortMain.java`, which is a lightly modified version of the `Main` class from Project 1. It works more or less like Project 1: it generates some data, and sends it to the sort algorithm you have provided. (The `<sortType>` argument has been removed, since we only have one sort now - and so has the `million` argument.) As in Project 1, this class can only tell whether you sorted correctly - although the TAs will check your implementation to make sure you actually **attempted** Heapsort.

Second, I have provided `Proj02_PriorityQueueMain.java`, which drives automatic testing of your heap (being used as a priority queue). In this class, we will randomly insert and delete values in your Max Heap. It validates that you return the correct values when we call `removeMax()`, but doesn't validate **how** you do this. (The TAs will look at your code, of course!)

Third, I will provide some testcases (named `Test*.java`), which will perform insert, delete, and build-max-heap operations using your class. But they will also regularly call `dump()` - and when this happens, you will print out the current contents of your array. Each file will be matched with a `.out` file, which gives the **exact output** which the testcase must produce (including your dump output). This will be used to automatically check if you are bubbling up and down in the proper manner, in all cases.

1.3 `.dot` File

As with Project 1, both of your classes must support a `debug` argument in each constructor - and if this argument is `true`, then your code must produce debug output. As with Project 1, the format of this debug output is up to you - but this time, it **must** include the ability to build a picture of the current heap using the `dot` tool.

Generate lots of these files in your debug output - if you generate enough, you can basically build an animation of your heap as it changes over time - and that's **very** useful for debug.

Use this feature early! When I was debugging my own implementation (yes, even the instructor has bugs), the very first thing that I wrote was code to generate this file. I then used it **extensively** throughout my debug - and frankly, I didn't really need any other debug code!

2 Style Requirements

2.1 No Global Variables

Hopefully, this isn't surprising to you. You may have any number of instance variables in your classes (in fact, that's **necessary!**) but global variables are **forbidden.**

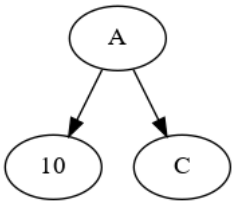
3 The `.dot` File Format

In Computer Science and Mathematics, a "graph" is a structure which contains a set of vertices (sometimes called "nodes") which are connected by edges. A binary tree, for instance, is simply a special type of graph - one where every vertex has (up to) two child edges pointed at other nodes (and a few other requirements).

The `.dot` file format describes a graph as a set of vertices and edges. A very basic dot file looks like this:

```
digraph
{
    A;         // this is a vertex named 'A'
    10;        // this is a vertex named '10'
    C;
    A -> 10;   // this is an edge going from A to 10
    A -> C;
}
```

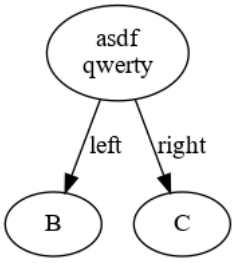
The code above produces the following picture:



The `.dot` file format allows you to pass arguments to each vertex or edge. There are many different arguments allowed; however, the simplest one is `label`, which allows you to assign the printed label to a vertex or edge. The label name is a string constant (exactly like C or Java), and can include special characters like newlines. Look at the next example:

```
digraph
{
    A [label="asdf\nqwerty"];
    B;
    C;
    A -> B [label="left"];
    A -> C [label="right"];
}
```

This produces the following picture:



4 Installing and Running `dot`

`dot` is a command-line tool which turns `.dot` files into graphics files (typically, `.png`).

4.1 Installation

`dot` is part of the Graphviz package (`https://www.graphviz.org/`). It is installed on Lectura and various department machines; the grading script will run it (if it is installed on your machine) as part of the testing process.

You can install Graphviz on your own machine, so that the grading script can test it locally; however, you will have to make sure that `dot` is in the `PATH` used by your shell.

Since I use Cygwin for most of my work (an open-source UNIX that runs inside Windows), I need help getting install instructions for various OSes! So far, people have posted on Piazza for the following OSes (and I've copied those instructions to the bottom of this spec):

- MacOS (Homebrew)

I'd also love instructions for:

- Windows 10 (UNIX subsystem)
- Linux (various distros)

4.2 Running `dot` By Hand

If you have `dot` installed on your machine, you can convert a `.dot` file to a `.png` using the following command:

```
dot -Tpng myFile.dot -o myFile.png
```

4.3 My `Makefile`

I have provided a `Makefile` which you may download if you wish. (Download it. Don't just copy-paste the contents, because `make` is picky about tabs vs. spaces.)

This makefile will automatically find any `.dot` files in the current directory, and call `dot` to convert every one of them to a matching `.png` file. (Take a look, it's amazing how easy this is to do!)

When I was testing my code, my basic testing process was:

- Make a small change to the code
- Compile everything with `javac *.java`
- Run one of the `Main` classes, passing it the `debug` option so that it would generate lots of `.dot` files.
- Run `make` to turn all of the `.dot` files into `.png` files.
- View the updated graphs.

4.4 Cygwin

I'm a huge fan of Cygwin (`http://cygwin.org/`) - although you have to be a **pretty major geek** to like it. Basically, you can think of Cygwin as a UNIX layer that runs **inside** Windows. I use it for my shell, for my editor (vi), for `ssh`-ing to Lectura and other machines, and many other common tasks.

If you want to play around with it, you're welcome to try it out - from the URL above, click the "Install Cygwin" link. Run the launcher it provides you; it will install a good default set of packages, which will give you a terminal and a

basic set of tools. If you run the launcher again later, you can add new packages (or install updates).

To get the `dot` tool, install the `graphviz` package. Other packages that you might find handy are (some of these might be installed by default, I don't recall):

- `xterm`

  Simple X windows terminal.

- `openssh`

  Provides the `ssh` and `scp` commands.

- `gcc`

- `make`

- `vim` and/or `emacs` (and/or `nano`)

- `screen`

  A tool that lets you have multiple virtual terminals inside the same shell window - and to switch between them trivially. (I'm currently running 8 `bah` shells inside a single window.)

  A tutorial: `https://www.mattcutts.com/blog/a-quick-tutorial-on-screen/`

NOTE: To use `java` and `javac` on Cygwin, I use a normal JDK install (`http://java.oracle.com/`), and then add the proper directory to my PATH variable:
`https://stackoverflow.com/questions/4918830/how-can-i-set-my-cygwin-path-to-find-javac`

4.5 Workaround 1: Download `.png` Files from Lectura

If you don't have Graphviz on your own machine, you can run the grading script on Lectura, then **copy the `.png` files back** to your own computer. You can then look at them on your own computer.

It's slow, but it works.

4.6 Workaround 2: Online Converter

Another option is to copy the text of the `.dot` file into the form on the following website:
`http://sandbox.kidstrythisathome.com/erdos/`

This website automatically runs `dot` for you, and will show you the picutre that results. However, this is a little slow (as you have to cut-n-paste by hand), and also it limits the size of the allowed graphs.

5 Java Implementation Overview

5.1 `Proj02_HeapSort`

This class must implement the `Proj01_Sort` interface (this is exactly the same interface as Project 1). The `sort()` method, of course, takes an array of `Comparable` and sorts it in place; it returns `void`[1].

Depending on your implementation, this class might not have a lot of code; it may simply use the `MaxHeap` class to build and then modify a heap. However, it must obey the following restrictions:

- You may create a new `MaxHeap` object - but you **must not** allocate a new array for any reason!

  (Allocating an `ArrayList` - or any other container - counts as allocating a new array.)

- I do not specify **how** you share code between this class and the `MaxHeap` class, but you **must not** have multiple copies of the "bubble up" or "bubble down" code.

  You may have **one** copy of the "bubble up" code, in either the `HeapSort` or `MaxHeap` class. You likewise may have one copy of the "bubble down" code

  (Of course, you don't have to make the "bubble up" and "bubble down" the same; they can be two different functions. But no more that **one of each.**)

- This class is required to have a `debug` parameter in its constructor; if we call `sort()` after setting `debug=true`, then you must generate debug output (including `.dot` files where useful). However, it's not strictly necessary that this code be written inside the `HeapSort` class. If `MaxHeap` does enough debug, then that's OK.

5.2 `Proj02_MaxHeap`

This class implements a max heap. It stores objects of type `Comparable`, and must use an array (**not an `ArrayList` or other container class!**) as its internal storage.

`MaxHeap` must support (at least) two constructors, which are called by my code:

- `Proj02_MaxHeap(boolean debug)`

  This works like the constructors from Project 1: it sets the debug flag. If this constructor pre-allocates an array to hold future data items (which I encourage), the array must have **no more than 4 items.**[2]

[1] Don't try to rewrite the interface! That makes the grading script not work!

[2] This is simply because I want you to be forced to test your array-expansion code a lot.

- `Proj02_MaxHeap(boolean debug, Comparable[] arr)`

  This sets the `debug` flag, but also initializes the heap with some input data. Build the max heap using the algorithm we've discussed in class (the $O(n)$ version). This method **must not** allocate any other array or data structure; it must build the heap inside the array given.

  You may assume that the array is not empty.

`MaxHeap` must implement the following methods, which are called by my test code:

- `insert(Comparable)`

  Inserts a new value into the heap. Use the algorithm that we've discussed in class (and in the slides).

  If the array is already full, your code must re-allocate the array and expand it. It must always expand the length by some fixed multiple (I suggest 2x, but it's up to you).

- `Comparable removeMax()`

  Removes the maximum value from the heap, using the algorithm that we've discussed in class and in the slides.

  This method **must not** reallocate (or discard) the array.

- `dump(PrintWriter)`

  Prints out the current contents of the heap[3].

  If there are no elements in the heap, print out a blank line (no whitespace).

  If there are one or more elements in the heap, print them out on one line, with exactly one space between each.

- Debug Code.

  One or both of your two classes (probably `MaxHeap`) must include which generates a `.dot` file. When `debug` is set (for the sort, or the heap itself), then you must generate `.dot` files (probably several) as the heap changes.

  If you want to also print out debug information to `System.out`, this is allowed but not required.

6 Base Code

Download all of the files from the project directory

`http://lecturer-russ.appspot.com/classes/cs345/fall18/projects/proj02/`

If you want to access any of the files from Lectura, you can also find a mirror of the class website (on any department computer) at:

`/home/russelll/cs345f18_website/`

[3] Just the actual contents of the heap - **not** including the big "empty space" at the end of the array

7 A Note About Grading

Your code will be tested automatically. Therefore, your code must:

- Use exactly the filenames that we specify (remember that names are case sensitive).
- **Not** use any other files (unless allowed by the project spec) - since our grading script won't know to use them.
- Follow the spec precisely (don't change any names, or edit the files I give you, unless the spec says to do so).
- (In projects that require output) match the required output **exactly!** Any extra spaces, blank lines misspelled words, etc. will cause the testcase to fail.

To make it easy to check, I have provided the grading script. I **strongly recommend** that you download the grading script and all of the testcases, and use them to test your code from the beginning. You want to detect any problems early on!

7.1 Testcases

You can find a set of testcases for this project at
`http://lecturer-russ.appspot.com/classes/cs345/fall18/projects/proj02/`

See the descriptions above for the precise testcase format, and also for information about how to run the two test driver classes.

7.2 Other Testcases

For many projects, we will have "secret testcases," which are additional testcases that we do not publish until after the solutions have been posted. These may cover corner cases not covered by the basic testcase, or may simply provide additional testing. **You are encouraged to write testcaes of your own, in order to better test your code.** You are also encouraged to share your testcases on Piazza!

7.3 Automatic Testing

We have provided a testing script (in the same directory), named `grade_proj02`. Place this script, all of the testcase files, and your program files in the same directory. (I recommend that you do this on Lectura, or a similar department machine. It **might** also work on your Mac, but no promises!)

8 Turning in Your Solution

Turn in the following files:

```
Proj02_HeapSort.java
Proj02_MaxHeap.java
```

You must turn in your code using D2L, using the Assignment folder for this project. Turn in only your program; do not turn in any testcases or other files.

9 Student-Submitted `Graphviz` Install Instructions

The following instructions were submitted to the class through Piazza, during this semester or a previous one. I cannot vouch for them, their correctness, or the reliability of the software they suggest. But I've included them in case it's helpful to you.

9.1 Mac OS/Homebrew

Author: Rajeev Ram (with edits by Peter Mahon), Fall 18

Installing Graphviz on Mac is straight-forward. I took the Homebrew route. If you don't have Homebrew installed on your machine, all you need to do is open Terminal and enter the command (per their website):

```
/usr/bin/ruby -e "$(curl -fsSL
   https://raw.githubusercontent.com/Homebrew/install/master/install)"
```

After Homebrew installs, enter the command

```
brew install graphviz
```

The process takes a while to complete (5 min or so), but it's really straight-forward and requires minimal effort. If your Terminal window is not doing anything and doesn't have the typical prompt/won't let you type anything, it's either downloading or installing something, so wait it out.