Finding Prime Numbers

Cosc 241 Assignment 1

Due: 4pm Tuesday April 9th 2013

The prime numbers 2, 3, 5, 7, 11, 13, 17,... are exceedingly important in computer science because of the part they play in error-correcting codes and in data encryption. This assignment is about algorithms to find all the prime numbers up to some limit, N. We give you Algorithms 1 and 3 and you yourself must design Algorithm 2 by making a modification to Algorithm 1.

You must implement all three algorithms, run them with sample values of N, and determine empirically which are the best and worst.

Hand in your programs along with some graphs of execution time plotted against values of N so that a good comparison of the algorithms can be made.

Algorithm 1

In this algorithm you consider every number from 2 up to N. If it is prime you include it in the list. To test whether a number m is a prime number you simply divide it by each of the numbers 2, 3,..., m-1; if any of these divisions is without remainder, then m is not prime, otherwise it is.

Algorithm 2

Modify Algorithm 1 to make the test for primality more efficient.

Algorithm 3

This algorithm is a little more tricky. It was discovered by the ancient Greeks and nowadays we attribute it to Eratosthenes of Cyrene (276BC - 194BC). You begin with an array P of booleans with one position for each index from 2 up to n (in Java arrays begin at index 0 but that just means that we won't actually use positions 0 and 1). You fill this array with the value 'true' at each position, except for $P[0]$ and $P[1]$ which are set to 'false'.

The next step is repeated until we are finished.

We look for the next position in the array which has a 'true' value (this will be position 2 initially, of course). Suppose this position is position t. Then we make $p[2t]$, $p[3t]$, $p[4t]$,... all 'false' and we add t to our list of primes. That's it! Carry out a few hand calculations to ensure you understand what is going on.

Input and output

Your program should take two command-line arguments (the algorithm number, and N) and output the primes using `System.out.println` and the timing data using `System.err.println` - so you could run your program like this:

```
java -jar Primes.jar 1 17
```

and get output like this:

```
2
3
5
7
11
13
17
1
```

The first 7 numbers are prime numbers ≤ 17 produced using algorithm 1, the last number is the time taken to run the algorithm. Note that because the output is going to two different output streams (System.out and System.err) it can sometimes get interleaved. So, in the example above the 1 might not be the last number printed.

Calculating the time taken

You can use `System.currentTimeMillis()` to work out the time taken to perform an algorithm like this:

```
long startTime = System.currentTimeMillis();
// do something that you want to time.  e.g. calculating primes
long endTime = System.currentTimeMillis();
System.err.print(endTime - startTime);    //prints time taken
```

Graphing the algorithm execution time

To get a good idea of how each of the algorithms perform for increasing values of N you should produce some graphs which plot N against execution time. To obtain the values you want to graph you could run your program a number of times using different values of N by using a shell script, for example:

```
for ((i=0; i<100000; i+=10000)) do
   printf "$i\t"
   java -jar Primes.jar 2 $i >/dev/null
done
```

This would run algorithm 2, and produce output like this:

```
0       0
10000   30
20000   56
30000   82
40000   98
50000   111
60000   124
70000   135
80000   174
90000   181
```

You could direct the output of the script above to a file by adding

```
&>| alg-2.txt
```

at the end of the line containing `done`. This basically says redirect the output to a file called `alg-2.txt`, overwriting it if necessary. The `&` is included before the `>` to indicate that both *stderr* and *stdout* should be redirected into the file.

It is up to you how you produce your graphs. There is a command line program installed on the Linux machines called `gnuplot` which you might find useful. Using the `alg-2.txt` file shown above you could produce a simple graph with the following command:

```
gnuplot -e 'set term pdf; set output "alg-2.pdf"; plot "alg-2.txt" with linespoints'
```

Submission

You should have a maximum of three graphs to submit with this assignment. Please make sure that they are PDFs.

Use the package name **asgn1** for this assignment. You need to submit your work (including the PDFs of your graphs) electronically by 4pm on the due date. Your program should compile using the **ant** build file which has been provided:

```
/home/cshome/coursework/241/files/build.xml
```

See lab 9 for more details about using *ant* and the changes you need to make to the build file. To submit your files change into the top level directory of your assignment and run the command:

```
ant submit
```

Check that the list of submitted files is correct, and that it includes your graphs. Marks will be deducted for late, or incorrectly named assignments and files (this includes incorrect case for your source files).

Any assignments submitted after the due date and time will lose marks at a rate of 10% per day late.

This assignment should be done individually. DO NOT work with other students, or ask the demonstrators for help. Treat it as a take-home exam. However, feel free to come and discuss any questions you may have with Iain Hewson.

All programs will be checked for similarity.

Marking

This assignment is worth 9% of your final mark for Cosc 241. It is possible to get full marks. In order to do this you must write correct, well-commented code which meets the specifications and produce sensible informative graphs which show the relative performance of the three algorithms in terms of execution time.

Marks are awarded for your program based on both implementation and style (although it should be noted that it is very bad to style to have an implementation that doesn't work).

Allocation of marks	
Implementation	4
Style/Readability	3.5
Comparison graphs	1.5
Total	9

In order to maximise your marks please take note of the following points:

- Your code should compile without errors or warnings using ant.
  If your code does not compile, it is a considered to be a very, very bad thing!
- Your program should use good Java layout as demonstrated in the code provided for the third lab.
- Make sure each file is clearly commented.
- No line should be more than 80 characters long, to avoid wrapping when printing.
- Most of your comments should be in your method headers. A method header should include:
  - A description of what the method does.
  - A description of all the parameters passed to it.
  - A description of the return value if there is one.
  - Any special notes.

Part of this assignment involves you clarifying exactly what your program is required to do. Don't make assumptions, only to find out that they were incorrect when your assignment gets marked.

If you have any questions about this assignment, or the way it will be assessed, please see Iain or send an email to ihewson@cs.otago.ac.nz.