Assessed Coursework MA407

Algorithms and Computation

due: Monday 10 February 2020 at 14:00

Instructions

The tasks on the following pages describe your assessed coursework, which constitutes **25%** of the final mark for this course.

It is due on **Monday 10 February 2020 at 14:00** (start of week 4 of Lent Term 2020). The tasks ask you to complete up to **4** Java files in plain text format. Make sure that your files compile on LSE computers, without the loading of any external Java packages.

Submit the two files `Graph.java` and `Tree.java`, plus up to two additional files with further public classes that you may need, on the Moodle page for MA407 under the submission link in the Assessed Coursework section.

Make sure that each of your files contains your **candidate number** (not your student number; see below).

Please submit only a single final version of your files. Otherwise you risk that the wrong submission will be marked.

The deadline is sharp. Late work carries an automatic penalty of 5 deducted marks (out of 100) for every 24 hours that the coursework is late.

Submission of answers to this coursework is mandatory. Without any submission your mark for this course is incomplete, and you may not be eligible for the award of a degree.

The work should be yours only. **Plagiarism** is considered an assessment offence at the LSE and is, as an instance of academic misconduct, taken very seriously. In a case of suspected plagiarism, the Department will act according to the School's Regulations on Assessment Offences—Plagiarism. So only submit work that is solely your own. This also means that for developing your solutions you are not allowed to collaborate with other students or to ask other persons for help.

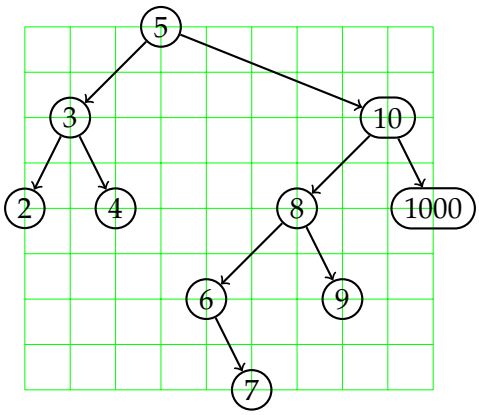
The contents of your work **must** remain **anonymous**, so do not write your name or student number in any of the program files. Instead, identify your work with your **candidate number**. Please insert your candidate number as a comment in the first line in each of your files. You can find your candidate number on 'LSE for You'.

Coursework MA407

Your task is to extend by various methods the data structure of a *binary search tree* from week 7. The main aim is to *nicely draw* the tree, as in the following example:

---

5 10 8 6 3 4 2 7 9 1000 are inserted



Let the program be called `Tree.java`, which when started with

```
java Tree 5 10 8 6 3 4 2 7 9 1000
```

could produce the following **text output** to create the above diagram, as we will explain:

```
\documentclass[a4paper,11pt]{article} %%%%%%%%%%% start of LaTeX file
\usepackage{mathpazo}
\usepackage{tikz}
\usetikzlibrary{shapes}
\oddsidemargin -0.54cm
\textwidth 17.0cm
\textheight 24cm
\topmargin -1.3cm
\parindent 0pt
\parskip 1ex
\pagestyle{empty}
\begin{document} %%%%%%%%%%% end of LaTeX preamble, start of text
\medskip\hrule\medskip
5 10 8 6 3 4 2 7 9 1000 are inserted

\begin{tikzpicture}[scale=0.600]
\draw [help lines, color=green] (1,-10) grid (10,-2); % creates help grid
\draw [thick] (10,-6) node[draw, rounded rectangle] (A) {1000};
\draw [thick] (8,-8) node[draw, rounded rectangle] (B) {9};
\draw [thick] (6,-10) node[draw, rounded rectangle] (C) {7};
\draw [thick] (5,-8) node[draw, rounded rectangle] (D) {6};
\draw [thick] (7,-6) node[draw, rounded rectangle] (E) {8};
\draw [thick] (9,-4) node[draw, rounded rectangle] (F) {10};
\draw [thick] (3,-6) node[draw, rounded rectangle] (A6) {4};
\draw [thick] (1,-6) node[draw, rounded rectangle] (CC6) {2};
\draw [thick] (2,-4) node[draw, rounded rectangle] (4_4) {3};
\draw [thick] (4,-2) node[draw, rounded rectangle] (abc) {5};
\draw [->, thick] (F) to (A);
\draw [->, thick] (E) to (B);
\draw [->, thick] (D) to (C);
\draw [->, thick] (E) to (D);
\draw [->, thick] (F) to (E);
\draw [->, thick] (abc) to (F);
\draw [->, thick] (4_4) to (A6);
\draw [->, thick] (4_4) to (CC6);
\draw [->, thick] (abc) to (4_4);
\end{tikzpicture}

\medskip\hrule\medskip
\end{document} %%%%%%%%%%% end of text and of LaTeX file
```

Generating a LaTeX file from Java by re-directing the standard output

The text above (also on Moodle as `test-latex.tex`) is a program in **LaTeX**, which is a typesetting program for mathematical text that you will be using for writing an MSc thesis (if this is part of your degree course). Your Java program will produce such a LaTeX file.

In a LaTeX file, anything from the % character onwards until the end of the line is ignored, so % starts a comment just like // does in a Java program. The first line above is therefore equivalent to `\documentclass[a4paper,11pt]{article}` and would be printed with the Java command `System.out.println("\\documentclass[a4paper,11pt]{article}");` where the double backslash \\ is needed because a single backslash introduces a special character (as in \n for "new line") where the special character is here the backslash itself.

The Java methods `System.out.print()` and `System.out.println()` print to standard output. When you start your program, you should **re-direct** the standard output to a file, for example named `a.tex`, with the re-direct symbol > as in

```
java Tree 5 10 8 6 3 4 2 7 9 1000 > a.tex
```

which will then send its output to the file `a.tex`. This file could be named in any way, such as `myoutput.tex`, but should end in `.tex` to show that it is a LaTeX file.

A redirection such as `java Tree 3 1 2 4 > a.tex` normally requires that you start the Java program from the **command line** (it may also work in the "Interactions Pane" of DrJava). Get a command line to work for you, which is the **terminal** program on Mac computers and either the **cmd** prompt or an environment such as **cygwin** (which you have to install) on Windows computers. You have to make sure to be in the correct directory (i.e., folder) which contains your Java file and compiled Java class. Starting programs from the command line is old-fashioned but reliable and fast. You should learn the minimal things about it for this coursework. On the command line, you typically can access previous commands with the up-arrow key so that you do not have to type them again.

From the command line, you should also have access to the LaTeX program that translates your LaTeX file to a pdf file (again, do **not** use a LaTeX development environment for this because you will not edit the .tex file here, only view it and generate it automatically). The command `pdflatex a.tex` will compile your file `a.tex` into a file called `a.pdf` which contains the generated pdf file, assuming your LaTeX file contains no errors.

You can then **view** your pdf file with a pdf viewer, such as Acrobat Reader, or a web browser such as Chrome. This is done by clicking on the pdf file in your folder (on some systems, from the command line with `acroread a.pdf`). When you generate the .tex and .pdf file again with `java` and `pdflatex`, you have to re-load it in your pdf viewer; some pdf readers automatically re-load the pdf file when it has changed.

In summary, a typical program execution is

```
java Tree 3 1 2 4 > a.tex
pdflatex a.tex
```

and then look at `a.pdf` with your pdf viewer.

Structure of a LaTeX file and tikz commands

Copy the following minimal LaTeX file into a file such as `a.tex`; you can also generate it with the program `Minimal_latex.java` which is on Moodle. Then compile `a.tex` with pdflatex

to see the single sentence "Hello, this is my text" on a page with a page number 1 at the bottom in the resulting pdf file:

```
\documentclass{article}
\begin{document}
Hello, this is my text
\end{document}
```

In general, anything before `\begin{document}` (see the larger introductory example on page 2) contains general information such as the layout of a page and a number of LaTeX packages. This is called the LaTeX **preamble** which is the header for your LaTeX file (and which you should generate with a Java method dedicated to producing it). Similarly, the terminal line of your text should contain the line `\end{document}` (again generated with a separate Java method).

Any text between `\begin{document}` and `\end{document}` represents normal text or special LaTeX commands that will appear, formatted, on your pdf pages. Line breaks in your text will considered just as blanks, so if you want to start a new paragraph you should type a **blank line** as in the introductory example on page 2 above.

The three LaTeX commands `\medskip\hrule\medskip` generate a bit of vertical space, a horizontal rule that goes across the full text width of the page (in the above file defined to be 17 centimetres), and another vertical space (omit `\medskip` to see the effect).

We use the special `tikz` LaTeX "environment" to draw graphs, shown above between the lines `\begin{tikzpicture}` and `\end{tikzpicture}` . This environment uses an implicit `grid` of coordinates, with standard grid size 1 cm. With `\begin{tikzpicture}[scale=0.600]` it is changed to 0.6 cm for the length of a grid square. Above we see a number of `\draw` commands that end in a semicolon ";" and that are convenient to start on separate lines. The tikz command `\draw [help lines, color=green] (1,-10) grid (10,-2);` draws a green grid between the coordinate pairs $(1, -10)$ and $(10, -2)$. Coordinates can have floating-point values but we will only use integers. The coordinates are only relative to each other.

The command `\draw [thick] (10,-6) node[draw, rounded rectangle] (A) {1000};` draws at position $(10, -6)$ a text, here "1000", surrounded by a rounded rectangle (with thick lines), and gives it a label, here `(A)` . This label (between parentheses) has to identify the shape (here the rounded rectangle) **uniquely**; the label could also be the pair of coordinates itself, such as `(10,-6)` instead of `(A)` .

These labels are then used to draw **arrows** between the rounded rectangles as in the command `\draw [->, thick] (F) to (A);` where in this case both shapes `(F)` and `(A)` have to be **defined earlier** in the LaTeX file because otherwise pdflatex would interrupt with an error such as `! Package pgf Error: No shape named F is known.` (In that case you should continue the LaTeX compilation by typing `q` to terminate it; then pdflatex may or may not produce a pdf file from what it has seen so far.)

Hence, when you draw a graph via tikz commands with nodes and directed edges (as you are asked to below) make sure you define the nodes before the edges.

Apart from drawing a graph as a tikz figure, you can also print normal text. Remember to create a blank line to start a new paragraph. In addition, you can type out error messages as LaTeX comments, as in `% Error: unknown end node (1,1) of edge` which will be ignored in the LaTeX compilation (anywhere, including in or before the preamble) but give useful information because they are visible in the LaTeX file.

Your coursework

For the following questions (a)–(g), the percentage of marks assigned to the individual answers are listed with each question, and are for (a) 35%, (b) 5% (bonus question), (c) 20%, (d) 20%, (e) 15%, (f) 10%, (g) 5% (bonus question).

The two bonus questions (b) and (g) give 5% of marks each and are added to your marks, but the total will not exceed 100%, which you can also achieve by answering the remaining questions perfectly.

Please write elegant and straightforward code, without *needless* case distinctions, and appropriate separate methods for conceptually separate tasks, and take care of borderline cases (for example, empty input). Clarity and readability of your program is part of the assessment criteria. Write appropriate (concise) comments. If you want to add any special features to your program (not required), or if you can only give partial answers to some questions, *document* this so we can take notice.

(a) The Graph class (35% of marks)

Create a Java program `Graph.java` with a public class `Graph` which allows creating a graph that is then output as a LaTeX file as described above. It should have the following public methods:

`addNode` Add a node to the graph, given by a pair of *integer* coordinates, together with a label string (such as "1000" in the example above). If the same coordinate pair is given again, update the label (so do not store the same node twice). The coordinates can be positive or negative but should in absolute value not be larger than an integer constant `maxcoordinate`, which you should set to 100 at a central place. Reject nodes where a coordinate exceeds this value with a LaTeX comment (on its own output line) as in `% coordinate 190 in node (50,190) too large`

`addEdge` Add a directed edge to the graph, given by the two pairs of integer coordinates of the endpoints of the edge. These nodes have to be *present* already; if not, reject them with a suitable LaTeX comment.

`clear` Empty the node and edge list.

`outheader` Output the header of the LaTeX file, i.e., the preamble up to `\begin{document}`

`outfooter` Output the footer of the LaTeX file, here the single line `\end{document}`. You could prefix it with `\medskip\hrule` if you like the resulting output better.

`outgraph` Output the graph as a `tikz` figure as described, with a blank line before and after, and including a green grid at the beginning. (There may be several such figures in a single LaTeX file, hence the separate `outheader` and `outfooter` methods.)

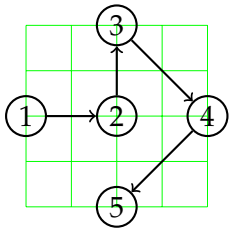
`outgrid` Output a grid between the smallest and largest coordinates that occur, in turn computed by a separate method.

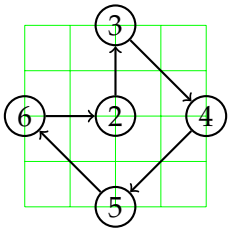
`main` This standard static method will only be used to **test** the `Graph` methods. Invoked with `java Graph` $x_1\ y_1\ x_2\ y_2\ \cdots\ x_n\ y_n$, it should create a path with edges $(x_1, y_1) \to (x_2, y_2), \ldots, (x_{n-1}, y_{n-1}) \to (x_n, y_n)$ from the sequence of pairs of coordinates given on the command line. The labels of the nodes are the numbers $1, \ldots, n$ (possibly overwritten when a node is used twice). For example,

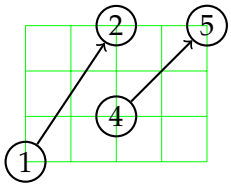
```
java Graph 0 0 2 0 2 2 4 0 2 -2
java Graph 0 0 2 0 2 2 4 0 2 -2 0 0
```

```
java Graph 1 1 3 4 10 200 3 2 5 4
```

(re-directed to a `.tex` file) generate, respectively, the graphs







As computed by the `outgrid` method, in the first two graphs the lower left corner is $(0, -2)$, and in the third graph it is $(1, 1)$.

(b) Grid adjustment in the Graph class (5% bonus marks)

Consider the `\textwidth 17cm` statement in the sample LaTeX file on page 2. This means that the text is printed on a width of 17 centimetres. An A4 sheet is 21 cm wide, so with equal margins (which we aim for) each margin is 2 cm wide. The standard left margin in LaTeX is 1 inch or 2.54 cm, so the additional margin (called `\oddsidemargin` above) should be `-0.54cm` as it is stated there to get a 2cm margin.

- Add a Java constant `textwidth` for the text width in centimetres which can be changed centrally, and from which the LaTeX layout parameters `\textwidth` and `\oddsidemargin` are correctly output.
- The standard grid size of 0.6 cm works well, but if the graph is too wide (as measured between the difference of smallest and largest horizontal coordinate) then it will go into or beyond the right margin. If the graph is too wide, adjust the grid size in the `[scale=0.600]` parameter depending on the textwidth, but not below 0.3, say, because otherwise the graph will look too dense.

Hint: When you print the scale or the textwidth, which is a double number in Java, use the `System.out.printf()` method. Suppose the double number is `a=0.42857142857`. Then `System.out.printf("%.3f", a)` will print it with three digits after the decimal point as `0.429`. You can also add further text to the formatting string: e.g., the output `[scale=0.429]` is created with `System.out.printf("[scale=%.3f]", a)` .

(c) Creating the binary search tree as a data structure (20% of marks)

As discussed in week 7, the Java program `BinarySearchTree.java` (on Moodle) describes a binary search tree. Re-name it as `Tree.java` and remove the comments before and after the program that describes the class. Also delete any methods that you won't need.

- The `main` method should read in the command-line arguments as integers and insert them into the tree. Print the numbers as they have been input to standard output (see the example below).
- Write a `printsorted` method that prints the numbers **in ascending order** to standard output. *Hint:* Use a straightforward recursion.
- Write a method `avgdepth` that computes the average depth (= level) of the nodes. The root has level 0. Output the number of stored keys as well. In the introductory example on page 2, the output is `average depth: 2.000, size 10`

Altogether, these outputs for the introductory example from page 2 should be

```
5 10 8 6 3 4 2 7 9 1000 are inserted

in sorted order: 2 3 4 5 6 7 8 9 10 1000

average depth: 2.000, size 10
```

where the blank lines are for LaTeX so they appear on separate lines in the pdf file.

(d) Draw the binary tree using the Graph class (20% of marks)

As part of `Tree.java`, write a method that draws a nice picture of the generated binary search tree as a LaTeX file. It should use the `Graph` class from (b) and its methods. The text that you have output in part (c) should come before the graph picture as text in the LaTeX file and later pdf file.

The initial example on page 2 shows you good choices of the coordinates for placing the nodes. The coordinates can be nicely computed with a `recursive method` applied to the tree, where a (recursively defined) `width` method is useful to count the children in a subtree, and it is useful to know whether that is a left or right subtree, to determine the position of the node relative to the parent node. You may possibly use as a feature the behaviour of `addEdge` to not draw an edge if an endpoint is missing (but then should explain where you use this).

The input to the program should be the command-line arguments as numbers to be inserted into the tree, like `java Tree 5 10 8 6 3 4 2 7 9 1000` to generate the picture on page 2.

(e) Generating a random permutation for tree elements (15% of marks)

For this part (e) and later (f) and (g), we will use a **different** interpretation of the command-line arguments if there are at most three of them (because binary trees with up to three elements are not very interesting anyway). Namely, the possible calls will be

- `java Tree -1` (the first number is negative), which tests the generation of a random permutation, see below;
- `java Tree` n where $n \geq 0$, which generates a random permutation of the numbers $1, \ldots, n$, and inserts them into the tree and displays it as before;
- `java Tree` n d, which generates n random numbers to be inserted, and subsequently replaces d further numbers;
- `java Tree` n d r, which repeats this r times and prints a table with r columns to compare the resulting average tree depths, in bonus question (g).

This question (e) asks the following: As part of the `Tree` class, write a method declared as `int[] randomperm(int n)` that generates an array whose n entries form a random permutation $a_1, a_2, \ldots, a_n$ of the numbers $1, 2, \ldots, n$ (the array size and array indices should follow the usual Java convention, with indices starting at 0). This method should use at most n calls to the method `Math.random` which generates a `double` number between 0.0 and 1.0 (exclusive). *Hint:* In each step, select the next element of the permutation randomly from a suitably represented set of numbers, and delete that number from the set.

Write a `randomtest` method that is used if the first command-line argument to `java Tree` is

a negative integer. It should output, say, 20 random permutations for $n = 3$ to see if your `randomperm` method does not accidentally avoid certain permutations.

In addition, **argue** with some text (as comment in your code) why your `randomperm` method is **correct**.

When called with `java Tree` n , your program should with `randomperm` generate a random permutation $a_1, a_2, \ldots, a_n$ and insert these numbers into the tree, and display it as before as if you had typed the numbers explicitly. A sample output is shown in part (f) below. If there is no command-line argument, do the same with default $n = 20$.

(f) Alternating random deletions and insertions (10% of marks)

When the program is started with `java Tree` n d, let $d = n$ if $d > n$. Then, generate a random permutation $a_1, a_2, \ldots, a_n, \ldots, a_{n+d}$, of $\{1, 2, \ldots, n + d\}$, which are the numbers to be inserted into the tree. However, only the first n of these numbers $a_1, a_2, \ldots, a_n$ are to be inserted at the beginning. This is what the program should do first. Then, compute and output the average depth of the tree nodes, and draw the tree. (So far, the program does nothing new.)

Next, compute another random permutation $q_1, q_2, \ldots, q_n$ of $\{1, 2, \ldots, n\}$. The first d of these numbers, $q_1, q_2, \ldots, q_d$, are the *indices* of the numbers to be deleted. The program should now proceed, for each $i = 1, 2, \ldots, d$, to first

(i) delete a_{q_i} from the tree, and then

(ii) insert a_{n+i} into the tree.

So these are d alternate delete/insert operations applied to random elements in the tree. If $d = n$, then all n numbers $a_1, a_2, \ldots, a_n$ are thereby replaced by the second set of n numbers $a_{n+1}, a_{n+2}, \ldots, a_{2n}$.

After these d alternate delete/insert operations (which should be printed), give the new average depth of the tree nodes, and draw the tree again.

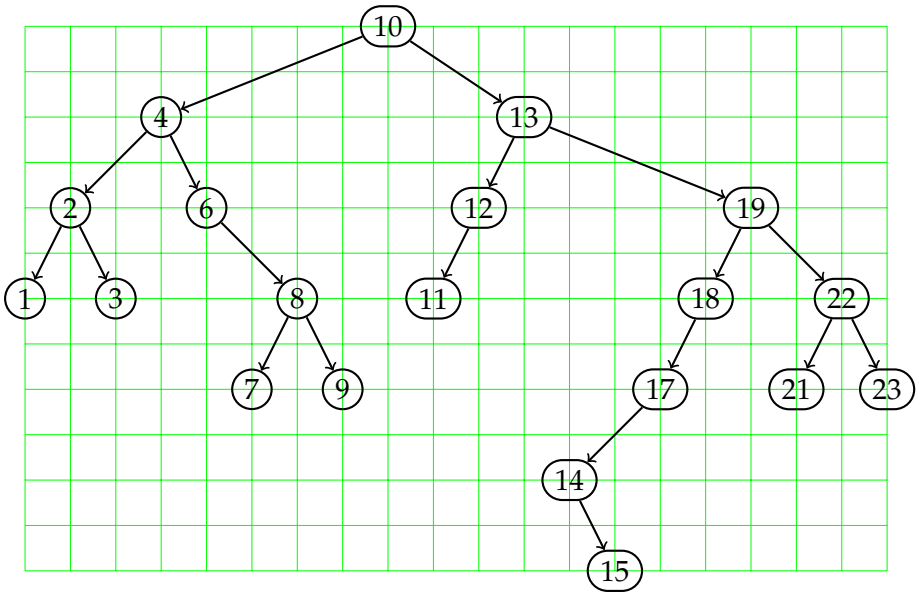
The following is an example when started with `java Tree 20 3`. First, it creates the following text output (as part of the LaTeX file, after the preamble):

```
10 4 13 19 22 18 2 3 1 21 17 12 14 11 6 8 15 23 9 7 are inserted

in sorted order: 1 2 3 4 6 7 8 9 10 11 12 13 14 15 17 18 19 21 22 23

average depth: 2.950, size 20
```

This means that it generates the search tree from 20 random numbers (out of 23, where the numbers 5, 16, and 20 are missing as can be seen from the sorted output). Then it nicely draws the tree, as follows. The subsequently described deletions and insertions mean that first 3 is deleted, then 16 inserted, then 13 deleted, then 5 inserted, then 17 deleted, then 20 inserted.



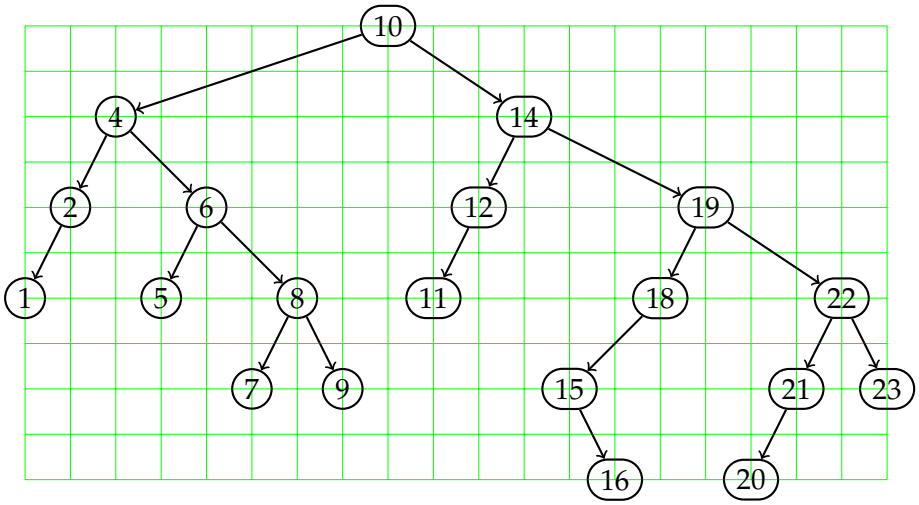
— alternate delete/insert: —

3 13 17 are deleted

16 5 20 are newly inserted

in sorted order: 1 2 4 5 6 7 8 9 10 11 12 14 15 16 18 19 20 21 22 23

new average depth: 2.900, size 20



(g) Testing of alternate deletions/insertions (5% bonus marks)

The effect of alternate deletions and insertions into a binary search tree is not well understood theoretically.

In this question, interpret three command-line parameters as in `java Tree` n d r as follows, with n and d as in (f). The third parameter r is a number of repetitions (so if it is missing as in (f), let $r = 1$). If $r > 1$, draw the tree only at the **first** iteration (or omit the tree drawing altogether). Then your program should display a nice table, which after `java Tree 40 20 10` could look like

Old depth:	5.38	5.60	4.35	4.83	5.03	4.78	6.13	4.10	4.38	5.28
New depth:	5.05	4.95	4.13	4.58	4.53	4.60	6.10	4.15	4.43	5.50

which in LaTeX is

```
\begin{tabular}{|l|c|c|c|c|c|c|c|c|c|}
\hline
Old depth:& 5.38& 5.60& 4.35& 4.83& 5.03& 4.78& 6.13& 4.10& 4.38& 5.28\\
\hline
New depth:& 5.05& 4.95& 4.13& 4.58& 4.53& 4.60& 6.10& 4.15& 4.43& 5.50\\
\hline
\end{tabular}
```

A specific tree would only be drawn for 4 or more numbers given on the command line. You can still draw trees with $n = 0, 1, 2, 3$ elements, by generating them randomly with `java Tree` n.

To repeat, correct answers to the two bonus questions (b) and (g) give 5% of marks each and are added to your marks. However, the total will not exceed 100%, which you can also achieve by answering the remaining questions perfectly.

We hope you enjoy this project!

END of assessed coursework for 2019/20 Good luck!