Practical Algorithms

University of Glasgow – 2024/25

Assessed Exercise 2

Submission deadline: Thursday, 12 December 2024, 20:00

Note: *This exercise is worth a total of 100 points and counts for 10% of the total mark for this course.*

Submission Instructions

Your solution must be submitted online on **Moodle**. It should contain *only* **three files**:

1. A PDF file named `Name_MatrNum.pdf`, where `Name` is your last name and `MatrNum` your matriculation number. You should write all your answers in this PDF file, except your Python source code for Questions 1(b) and 2(d), which should be provided in the following files:

2. `ValidParentheses.py`: your Python implementation for Question 1(b)

3. `BSTHeightHistogram.py`: your Python implementation for Question 2(d)

Your Python scripts should be able to be executed (without errors) in a standard Unix shell, by running the command

```
python file_name.py
```

assuming a standard, relatively modern Python installation (i.e. version 3.11 and above).

No other operating-system dependent setup should be assumed, and your execution should not rely on any special `PATH` variable configuration, IDEs, or dependencies. In a nutshell, *we should be able to easily run your scripts!*

Finally, you should not make use of any special Python modules, and you may only import the classes that we explicitly allow you to in the statement of Question 2(d).

Question 1 *(40 points)*

You want to design an algorithm that solves the following problem: as input you are given a string that may contain *only* the parenthesis characters

`'('`, `')'`, `'{'`, `'}'`, `'['` or `']'`,

and you are asked to determine whether the input expression is "valid", i.e., if the parentheses are properly balanced. For example, expression `'[{()}]()'` is valid, but expressions `'[{()]()'`, `'[{{()})]'` and `')('` are *not* valid.

(a) Discuss how you can use a stack data structure to solve your problem.

(20 points)

(b) Implement your algorithm in Python. When run, your code should prompt the user to input the string of parentheses; then, it should correctly output `"The expression is valid"` or `"The expression is not valid"`.

You may assume that your input string will contain only parenthesis characters, and no other special characters. That is, your code is expected to be able to handle inputs like `'[{{()}}]'`, but you do not need to worry about dealing with inputs like `'[,{,{,(,),},},]'` or `'[ { { ( ) } } ]'`.

(20 points)

Question 2 *(60 points)*

A binary tree is called *perfectly balanced*, if the left and right sub-trees of all its nodes have *exactly* the same height.

(a) Prove that any (non-empty) perfectly balanced binary tree of height h has exactly $2^{h+1} - 1$ nodes. *Hint: Try using mathematical induction.*

(13 points)

(b) Give *two* different orderings in which we can enter the values $\{1, 2, \ldots, 7\}$ in a (initially empty) binary search tree (BST) so that in the end it is perfectly balanced. For one of these input sequences (whichever you like), draw *all* intermediate BSTs that arise after each insertion (that is, you have to draw seven BSTs).

(10 points)

(c) Draw *two* different BSTs whose in-order traversal outputs the sequence $1, 2, 3, 4, 5$.

(7 points)

(d) Implement the following algorithm in Python: you ask the user to input a positive integer n. Then, for each of the $n!$ many different permutations of the values $1, 2, \ldots, n$, you compute the height of the BST that arises if we insert, at an initially empty BST, that specific sequence of elements. Then, you print a histogram with the frequencies of the different heights than can arise, as well as the average height.

Your output should be in the following form (here given for $n = 6$):

```
Height  Frequency
----------------
0       0
1       0
2       80
3       400
4       208
5       32

Average height of BSTs: 3.2666666666666666
```

That is, for each possible value of the height $h = 0, 1, \ldots, n-1$, at the column "Frequency" we write down how many of the $n!$ permutations resulted in a BST of height equal to h.

For your Python implementation, feel free to import the class `permutations` from Python's module `itertools`, which you can use to iterate over permutations. In other words, you are allowed to use the following call at the preamble of your Python code

```
from itertools import permutations
```

but you are *not* allowed to import any other additional functionality or any other modules.

Note on running time: Since the total number of permutations $n!$ grows *exponentially* as a function of parameter n, naturally you will observe that your script slows down (dramatically) for larger values of n. This is unavoidable, given the requested implementation, and you don't need to worry too much about optimizing your running time: as long as your script runs within a reasonable amount of time (let's say, 30 seconds), for up to $n = 10$ you are good to go!

(30 points)