Algorithms & Data Structures 2018/19 Coursework

Matthew Johnson & Rob Powell

Hand in by Friday 18 January 2019 at 2pm in DUO.

For each question, read carefully what you are required to submit. The files should be submitted together as a your_id.zip. It is simplest to use your CIS username as your_id as duo stamps your submissions with this anyway.

1. This question requires you to produce python code to add keys to a hash table. The approach is the same as in Question 3 of Practical 5 (see `adspractical5.pdf` and also `adspractical5solutions.py`); the crucial difference is the way that collisions are handled.

   You must create the code for two functions `hash_quadratic` and `hash_double` that add keys (we ignore the values) to a hash table of size 19 using the hash function defined by $h(k) = 6k + 3 \bmod 19$. In `hash_quadratic` collisions should be handled with quadratic probing. In `hash_double` collisions should be handled using Double Hashing using the secondary hash function $h'(k) = 11 - (k \bmod 11)$. The input to the functions is a list of keys (positive integers) to add to the hash table and the output is the contents of the table where the symbol "$-$" indicates a bucket in the array containing no data. You can assume that the input contains no duplicates. Again, study `adspractical5.pdf` and `adspractical5solutions.py` to see examples of the format of inputs and outputs.

   Your submission for this question should be a single file `your_id_q1.py` containing the two functions `hash_quadratic` and `hash_double`. The file should not include any `import` statement. These functions will be tested on a number of inputs. Full marks will be awarded if all tests are passed. Some sample tests are available in `q1test.py`. **[15 marks]**

2. Let k be 2, 3, or 4. The *k-child* of a positive integer is defined to be a number obtained by taking the kth power of each digit and adding them. For example, the 2-child of 19 is $1^2 + 9^2 = 82$, the 3-child of 74 is $7^3 + 4^3 = 407$ and the 4-child of 66 is $6^4 + 6^4 = 2592$. The *k-descendant sequence* of a positive integer begins with the integer and then each successive term is the k-child of the previous one. The sequence terminates if it reaches 1 or a term is repeated. Here is the 2-descendant sequence of 19.

   $$19 \to 82 \to 68 \to 100 \to 1$$

   Here is the 3-descendant sequence of 74.

   $$74 \to 407 \to 407$$

   Here is the 4-descendant sequence of 66.

   $$66 \to 2592 \to 2498 \to 7218 \to 6514 \to 2178 \to 6514$$

   A positive integer is *k-ephemeral* if its k-descendant sequence ends in 1, and otherwise it is *k-eternal*. So 19 is 2-ephemeral, 74 is 3-eternal and 66 is 4-eternal.

Write a function, using python, called `count_ephemeral(n1,n2,k)` which, given two positive integers n_1 and n_2, $n_1 < n_2$, and $k \in \{2,3,4\}$, returns the number of k-ephemeral numbers i, $n_1 \leq i < n_2$. Save the function in a file `your_id_q2.py`. This file is all that you need to submit for this question. The file should not include any `import` statement and should not contain more than 50 lines of code (excluding blank lines and comments).

Your function will be tested on a range of inputs. The maximum value of n_2 will be 10,000,000. Each test is single call to `count_ephemeral(n1,n2,k)`. The test is passed if the correct value is returned in under one minute. Three of the tests that will be used are available in `q2test.py`. 10 marks will be awarded if all tests are passed. Up to 4 marks will be awarded for a sensible algorithmic approach. Up to 4 marks will be awarded for an efficient implementation.

[18 marks]

3. A *sum-and-product expression* is a statement containing positive integers, brackets and the plus and times operators. Here are some examples.

$$(1+2) \times 3 + 4 \quad (1)$$
$$1 + 2 \times 3 + 4 \quad (2)$$
$$1 \times 2 + 3 + 4 \quad (3)$$

which of course evaluate to 13, 11 and 9 since $\times$ has precedence over $+$. A sum-and-product expression is said to be *good* if none of the pairs of brackets it contains are redundant; brackets are redundant if their removal does not affect the calculation that is done. The statements (1), (2) and (3) are good. The following three statements are not good since brackets can be removed to obtain, respectively, (1), (2) and (3) without changing the implied calculation.

$$((1+2)) \times 3 + 4$$
$$1 + (2 \times 3) + 4$$
$$1 \times 2 + (3+4)$$

Note that $(3+2) \times 1$ is good; removing the brackets does not affect the value obtained but it does affect the way the value is calculated — without the brackets the multiplication is done before the addition. Note also that in a good sum-and-product expression there might be many layers of nested brackets such as in

$$(1 + 2 \times 3 \times ((4+5) \times 6 + 7) + 8) \times 9$$

Write an algorithm that decides whether a sum-and-product expression is good. Ideally the input should be read just once, from left to right. You can assume that the input is a correct expression except that it might have redundant brackets. So brackets in the expression are each part of a matching pair — this does not need to be checked. You can also assume that that all operators are explicit: $(1+2)(3+4)$ is not a sum-and-product expression as the multiplication is implicit. You can either write pseudocode or edit and rename `q3.py` to create a python function called `good_expression` that returns True or False. The file contains some defined data structures that might be useful. Do not use any `import` statement. The file also contains a test function. Note that you can assume that the input is a string containing integers, brackets and the plus and times symbol (an asterisk). For simplicity, also assume that the integers are all single digits and positive and that there are no spaces. For this question, submit either `your_id_q3.py` or `your_id_q3.pdf` containing pseudocode. **[17 marks]**

Provide your written answers for questions 4, 5 and 6 in a single file `your_id_q456.pdf`. (Note that python files are also needed for Question 6.)

4. Prove or disprove each of the following statements. We will assume that $x > 0$, and all functions are asymptotically positive. That is, for some constant k, $f(x) > 0$ for all $x \geq k$. You will get 1 mark for correctly identifying whether the statement is True or False, and 1 mark for a correct argument.

   (a) $2x^4$ is $O(x^3 + 3x + 2)$. **[2 marks]**

   (b) $4x^3 + 2x^2 \cdot \log x + 1$ is $O(x^3)$. **[2 marks]**

   (c) $3x^2 + 7x + 1$ is $\omega(x \cdot \log x)$. **[2 marks]**

   (d) $x^2 + 4x$ is $\Omega(x \cdot \log x)$. **[2 marks]**

   (e) $f(x) + g(x)$ is $\Theta(f(x) \cdot g(x))$ **[2 marks]**

5. For each of the following recurrences, give an expression for the runtime $T(n)$ if the recurrence can be solved with the Master Theorem. Otherwise state why the Master Theorem cannot be applied. You should justify your answers.

   (a) $T(n) = 9T(n/3) + n^2$ **[3 marks]**

   (b) $T(n) = 4T(n/2) + 100n$ **[3 marks]**

   (c) $T(n) = 2^n T(n/2) + n^3$ **[3 marks]**

   (d) $T(n) = 3T(n/3) + c \cdot n$ **[3 marks]**

   (e) $T(n) = 0.99T(n/7) + 1/(n^2)$ **[3 marks]**

6. Consider the MergeSort algorithm as we have seen in lectures, but we want to change the base case and the order of the sorting.

   (a) Instead of recursively calling MergeSort until the list to be sorted has length 1, we will instead implement SelectionSort to sort any list of length 4 or less, and we want the algorithm to output a sorted list from largest to smallest. These algorithms are sometimes known as hybrid sorting algorithms. Provide python code that performs this hybrid MergeSort and SelectionSort algorithm on an input array A. We will also assume that the input array A contains unique elements, so no element appears more than once, and the length of A is a power of two. You should submit a single file called `your_id_q6a.py` **[15 marks]**

   (b) Give an example of a worst case input to this algorithm. That is an input that requires the most comparison operations, along with justification as to why that is the case. **[4 marks]**

   (c) Now we want to generalise the algorithm to allow a user to enter an input array A of any length, not just a power of two. Provide new Python code to perform this hybrid MergeSort and SelectionSort algorithm on inputs of any length. We will still assume that the input array A contains unique elements, so no element appears more than once. You should submit a single file called `your_id_q6c.py` **[6 marks]**