Algorithmics Exam 2021 - home assignment

Period: From 8.-17. January. Tasks are given on Friday, submissions by next Sunday night.

Rules: Exam is personal, no collaborations or even discussions with any other students, parents, colleagues etc. (other people), is not permitted. For code, it is preferred to provide concise self-implementations instead of relying on copy-pasted code. There are theoretical subtasks and practical subtasks. It is possible to pass the exam with theoretical tasks only, but it is highly advisable to do some coding as well. Write on the report your full name and study book code. Begin the report by a statement that you completed the exam on your own and cited/reported all materials that you consulted and used in your solutions. Please also keep track of time spent and indicate the used hours for each task. Probably unnecessary hint: develop and test first on small(er) data sizes before final experimentation on full scale. Necessary hint: make sure to assess your time limitations. Probably there is no need to complete all programming tasks in the hard way, think through and choose wisely. Note also that tasks may not have any single “most correct” answers. Goal is to demonstrate independent thinking.

Submission: via courses.cs.ut.ee - the PDF and Code (zip)

Task 1: Order statistics (10p)

Data 1: Generate Fibonacci numbers\(^1\) in required modulos (see below)
\[
\text{Fib}(0)=1; \quad \text{Fib}(1)=1; \quad \text{Fib}(n)=\text{Fib}(n-1)+\text{Fib}(n-2)
\]
Data 2: Generate random integers in the chosen modulos. Fix the seed, e.g. seed(2021).

Your task is to generate one billion values of unsigned positive integers in three different modulos and perform the rank order query from these values. It is not forbidden to first save them first and then query, but most likely it is far more efficient to handle the analysis “on the fly”. If in doubt, test.

**T1.1:** Describe a short pseudoalgorithm for finding i\(^{\text{th}}\) value in this data stream sequence. Discuss merits, restrictions and flaws.

**T1.2:** What is the time complexity of your general solution? (for n values generated, find the i\(^{\text{th}}\) value).

**T1.3:** Repeat 3 times, for each modulo \(2^m\) s.t. \(m=20, 32, 64\)

\[
\text{For Fibonacci numbers and 10 random sequences (seed(2021).seed(2030))}
\]
- What is the order i\(^{\text{th}}\) (i=100, 100,000 and 10,000,000) value and how many (other) equal values does it have? Report in 3 tables (by \(m\) of modulo \(2^m\)).

**T1.4:** Measure and report the speed for each of the calculations, assess the effect of \(m\) and \(i\) each time. Report the same tables as in T1.2, but now for the time spent.

\(^1\) Corrected the Fib(n) definition
T1.5: What is the actual time complexity based on the course material? Would that method be suitable in this case in practice? Justify.

T1.6: Propose an approach and solve at least one subtask “much faster” than a generic solution. How much faster is the proposed approach? What is the time complexity of your proposed approach? Describe your method.

Task 2: Text processing (10p)

Take English and Estonian words:
https://raw.githubusercontent.com/dwyl/english-words/master/words.txt
http://kodu.ut.ee/~fishel/dict/est-lemmas.txt (UTF8)
http://kodu.ut.ee/~fishel/dict/est-words.txt (UTF8)

T2.1: Use UNIX command-line tools (quickest hack) or write a small program by yourself:
Remove all words that contain “bad” characters /punctuation, numbers, etc/
Report the number of words and total number of characters in final files (e.g. run wc)
Report frequency count for all word lengths

You can also use simple sorting and counting the unique words type solution:
```
sort | uniq -c | sort -nr
```
Find words that occur in both languages, report the ten longest ones
Find palindromes (in each), report ten longest ones.
Find words that are reverse in one language vs the other, report ten longest examples.

When using command-line, keep track of command line, copy-paste these to report.

T2.2: For both languages, calculate for all letter pairs “before - after” frequency (count of words; possibly multiple times within each word). ABBBC - AB x2, AC , BB, BCx2. For every character pair, calculate the log ratio. Is the pseudo-count (+1) necessary?

```
log( (freq(a,b)+1) / (freq(b,a)+1) )
```

Which ten pairs have the strongest order-preference? Report counts and log-ratio.
For each, report 2-3 words in the “preferred” and the opposite order.
(Make a single table)
T2.3: Using that preferred order measure, find and report ten that score highest by “forward frequency”. Calculate the sum of such frequencies over all letter pairs within the entire word. Repeat the same for “reverse-frequency”. Use three variants - 1) sum all pairwise scores, 2) sum only the positive scores, 3) sum only the negative values. Test the effect of “normalising” the sums by length, e.g. by taking the average of the scores.

T2.4: For those interesting cases (from T2.3; or by some other choice of examples, if you did not do the T2.3.), identify ten most similar other words using simple edit distance (substitute, insert, delete). (Using agrep or other library or tool is ok)

T2.5: Propose a pseudalgorithm to find the longest substring that maximises (or minimises) the sum of such frequency scores for that substring. What would be the time complexity for that algorithm for a word of length n? Pick some examples and demonstrate that it would work correctly. (No actual need to code; but even better if computed over dictionary and reported the actual maximal example(s)).

Task 3: Deterministic acyclic finite-state automaton (DAFSA) (14p)

See: https://en.wikipedia.org/wiki/Deterministic_acyclicFINITE_STATE_Automaton

For either one of the languages:

T3.1: Draw a DAFSA of ten words (of length six or more characters) from the above English or Estonian words file, freely chosen by yourself. (Be clever in your choice of words)

T3.2: Describe an algorithm that would allow building a DAFSA (does not need to be the most efficient approach, better focus on ease of understanding)

T3.3: Take one of the words file and construct a DAFSA. State the size statistics for the DAFSA (number of words, lengths of words, and nr of states in DAFSA). Focus on correctness; if the speed is rate-limiting, use and report for less words.

T3.4: Based on the constructed DAFSA, find the ten substrings that form unique paths in DAFSA (such that only this word passes through these edges). Report the length and count statistics for unique single-word consecutive edges. Identify ten such longest substrings and the corresponding exact words that pass these edges.

T3.5: Use fgrep (multi-keyword search) to find from the dictionary the words that contain the same substrings. Report the count (wc) and time. E.g. run commands

head search.txt words.txt
time fgrep search.txt words.t | head
time fgrep search.txt words.t | wc
Task 4: BWT and indexing (6p)

**T4.1:** Decode the Burrows-Wheeler encoded text using "pen and pencil" by a linear time reconstruction algorithm. Briefly describe how the linear time algorithm works. The dot (\'.\') marks the last character of the sentence. There are 8 underscores \'_\' (one is in double). Alphabetic order is standard: \'.\', \'_\' and \'a'..'z'.

nnesmsysd_xx_anegln__dk_mths_wilaa_oiaikinile__coeem

**T4.2:** Draw a compact suffix tree of the decoded text. How many states does it have? Pay attention to the fact that it may appear large and tedious unless you think through how to represent it efficiently on paper. Simulate (on paper) the search for text 'exam' in the original text. Count the number of character comparisons required (describe the principle briefly and do not just report a number).

**T4.3:** Provide a suffix array of this decoded text. Simulate (on paper) the search for text 'exam' in the original text. Count the number of character comparisons required.