DELFT UNIVERSITY OF TECHNOLOGY
Faculty of Electrical Engineering, Mathematics and Computer Science

CSE2310 Algorithm Design
Digital test – part 1 of 2

December 6, 2021

- Usage of the book, notes or a calculator during this test is not allowed.
- This digital test contains 3 questions (worth a total of 10 points) contributing 10/20 to your grade for the digital test. The other digital test contributes the other 10/20 to your grade. Note that the final mark for this course also consists for $\frac{2}{5}$ of the unrounded score for the written exam (if both ≥ 5.0).
- Please answer any questions in clear English.
- Weblab contains public tests that give an example input/output for these exercises.
- If you have questions about the exam, you can use the WebLab discussions to ask them.
- The spec tests in WebLab **do not necessarily correspond with your grade**; your implementations will be manually graded.
- There is an API specification of the Java Platform available at `https://weblab.tudelft.nl/docs/java/17`.
- Please provide the **most efficient** implementation in terms of asymptotic time complexity. Providing a suboptimal implementation can lead to a subtraction of points.
- The material for this tests consists of module 1 and 2 of the course and the corresponding book chapters and lectures.
- Total number of pages (without this preamble): 1.

1. ($2\frac{1}{2}$ points) *You may ignore this paragraph of context if you so choose.* On the British television show *Taskmaster* comedians complete ridiculous tasks as well as they can. Their performance is judged only by the titular taskmaster. In a recent episode contestants were given a series of tasks to complete *as slowly as possible* while riding a bicycle. This makes you wonder, given a series of n tasks that all take 5 minutes to perform, how many could you do? In the middle of your thinking however, Alex Horne (the Taskmaster's assistant) tells you that each task i with a description d_i can only be completed at a specific time s_i.

   Given n tasks with a description d_i and start time s_i for every task, and the knowledge that completing a task takes 5 minutes, return the largest possible set of task descriptions you can complete.

2. Although Huffman's encoding works great for binary systems, we can shorten the encodings even more by using ternary (0, 1, 2). With a 0 for the left child, a 1 for the middle child and a 2 for the right child, we could have the encoding "021" now for "left child, right child, middle child".

   In order to make sure our tree remains full however we can run into issues when we have an even number of characters. In such cases we might have a situation where the tree is a bit off balance. Fortunately there is a straightforward solution: if there are an even number of characters, just add a node with a non-used character (for example character code 0, in Java: `(char) 0`), and a frequency of 0. Now the number of characters is odd and all should be well!

   (a) (1 point) Provide an implementation for the `decode` method that given an encoded ternary string and the ternary tree, returns the original message.

   (b) (3 points) Provide an implementation for the `buildTree` method that given n characters with a frequency f returns a *ternary* Huffman tree for the optimal prefix encoding.

3. ($3\frac{1}{2}$ points) On the topic of prefix encodings, we are interested in figuring out the largest prefix shared by a large set of strings. For example in the set { monkey, money, moe, monster} the largest common prefix would be *mo*. However if we add *phoenix* to the set, they no longer share any prefix, so the answer would be the empty string. Although a pseudopolynomial linear solution exists in $O(nm)$ time where n is the number of strings and m is the length of the longest string, the International Association of Parallel Prefix Processing has asked you to create a method using divide & conquer. This way they can put their many ~~humans~~ computing machines to work. Their requirement is that the run time complexity is still $O(nm)$ only now using a divide and conquer approach instead of an iterative approach to help in their parallelisation efforts.

   Implement a *recursive Divide & Conquer* algorithm that computes the longest common prefix given n strings.

End of exam questions