2011 Interleaving

Given two ordered lists of objects a new list can be formed by *interleaving* the two original lists. This is done by repeatedly removing the object at the start of either list and placing it at the end of the new list. This is continued until there are no objects left in the original lists. The rule for choosing (when both lists still contain objects) which list to take the object from, will vary depending on the algorithm.

For example, supposing our lists are [1, 2, 5, 8] and [3, 4, 6, 7, 8], three possible rules are:

- "Keep taking from the first list until it is empty" — our two lists combine to [1, 2, 5, 8, 3, 4, 6, 7, 8];
- "Keep taking a smallest object" — our lists combine to [1, 2, 3, 4, 5, 6, 7, 8, 8];
- "Alternate between the lists, starting with the first list" — our lists combine to [1, 3, 2, 4, 5, 6, 8, 7, 8].

Socks

Consider the following outline of a task:

A clothes-line holds a row of socks, each of which is either black or white. How many different arrangements are there with b black and w white socks?

Question 1
Rewrite this task in terms of interleaving.

The mathematical notation ${}_nC_r$ means the number of ways of choosing r objects from n objects. E.g. if we have the 4 objects a, b, c and d there are 6 different pairs of objects (a&b, a&c, a&d, b&c, b&d and c&d) so ${}_4C_2$ is 6. For the purpose of this question you do not need to know how this is calculated.

Question 2
Explain (without using the formula for ${}_nC_r$ if you do know it) why the number of different arrangements of socks is equal to ${}_{b+w}C_b$ and why that is equal to ${}_{b+w}C_w$.

Question 3
There are ${}_{b+w-1}C_{b-1}$ arrangements where the first sock is black (since there are only $b+w-1$ more socks on the clothes-line and only $b-1$ of them are black). Similarly there are ${}_{b+w-1}C_{w-1}$ arrangements where the first sock is white. Outline an algorithm, that uses these facts, to solve the task.

Traffic

Consider the following outline of a task:

Two lines of traffic merge into a single lane and, a little way down the road, this single lane again splits into two. There is no overtaking. Traffic is observed and recorded as it passes along the two original lines and again after it has split into two. Given four lists (the order of the traffic along the two original lanes and the two final lanes) determine if the lists are consistent; i.e. whether or not the recorded traffic in the final two lanes could have been generated from the first two lanes.

Question 4

Briefly outline an *efficient* algorithm for solving this problem. An *efficient* algorithm would rapidly solve the problem for large amounts of data.

Question 5

Briefly outline an *inefficient* algorithm for the problem. An *inefficient* algorithm would take a long time to generate solutions as the lists grow in size.

Question 6

Suppose you are producing test-data to mark the problem.

a) One objective when marking is to give marks to programs that are partially correct. Explain how you might generate test-data that good and bad programs might get correct.
b) A second objective is to try to write test-data that will catch out programs that are correct in some cases but wrong in others. Explain how you might generate test-data that only the best programs would get correct.

Reverse Interleaving

Suppose that we are interleaving a list of n elements with its reverse — i.e. the list that contains the same elements but in the opposite order. Regardless of how the interleaving takes place, the first n elements in the new list will be a permutation of the elements in the original list; similarly for the final n elements in the new list. We'll call this the *reverse interleave property*.

Question 7

Explain briefly why the *reverse interleave* property is true.

Question 8

Suppose that a list consisting of p copies of an n element sequence is interleaved with its reverse. What can you say about the resulting sequence?

… and finally

Question 9

Design a programming task that uses *interleaving* and / or the *reverse interleave property*. They may be used directly (e.g. within the task definition) or indirectly (e.g. required by the solution). You should explain the task, discuss one or more solutions for the task, and indicate how you would go about testing solutions.

Note:

- You do not need to "give a story" when explaining the task and you can refer directly to computer science concepts (e.g. interleaving) without explaining them.
- Do not forget to indicate appropriate bounds for your problem.
- When giving a solution you should discuss the algorithm; you do not need to give code.
- You should explain how you would design certain pieces of test data and what they are designed to test. You do not need to explicitly give test cases.

You can discuss anything that you feel relevant to your task, such as why certain decisions have been made and what relation the task might have to other tasks you have seen.