Smallest k sets

– Haskell lab assignment 1 –
– D7012E Declarative languages –

Håkan Jonsson
Luleå University of Technology, Sweden

May 5, 2023

1 Introduction

In this assignment your task is to write a program in Haskell that, given an integer $k > 0$ and a list ℓ containing n integers (that do not need to be unique), computes and prints a *smallest k set* of ℓ defined as follows:

A smallest k set of ℓ is a set $\{\ell_1, \ell_2, \ldots, \ell_k\}$ of k different non-empty sublists of ℓ such that for all other sets $\{\ell'_1, \ell'_2, \ldots, \ell'_k\}$ of k different non-empty sublists of ℓ,

$$\sum_{i=1}^{k} \sigma(\ell_i) \leq \sum_{i=1}^{k} \sigma(\ell'_i),$$

where $\sigma(\ell)$ computes the sum of the elements in ℓ[1]. Moreover, in the definition above, a *sublist* of a list is what we get by removing elements from the ends of the list.

Smallest k sets find application in data mining, especially efforts to find patterns in massive amounts of information ("Big Data").

2 How to proceed

Typically, the program computes all sublists and where in the input list they are located in terms of their start and end positions. Note that these positions start with 1, not 0.

[1] Empty lists have no size.

Then, the sublists, along with their sizes and oocations, are ordered ascendingly by size and the k first (which are then the smallest) are printed as shown below and in Section 5. Some details:

- There can be duplicate elements in the input list. This means a list could occur several times but at differenet locations. The input list `[9,1,2,3,8,1,2,3]`, for instance, contains the list `[1,2,3]` twice. The first occurance starts at position 2 and ends at 4 while the other starts at 6 and ends at 8.
- The input list could contain negative as well as positive integers.
- How sublists of equal sizes are ordered with respect to each other depends on the sorting algorithm used. As long as your program outputs one of the potentially many smallest k sets, also those different from the ones in this text, it is considered correct.
- Empty lists obviously have no non-empty sublists and therefore lacks smallest k sets. Given an empty list as input, you program should react by just calling `error` with a suitable error message.

3 Sublists and smallest k sets

Here is an example with a tabulation of all sublists of the list `[-1,2,-3,4,-5]` top-down from smallest to largest and with one sublist per row. In the table, columns `i` and `j` contain start and end positions of sublists. If we take out the k top-most rows, we get a smallest k set.

size	i	j	sublist
-5	5	5	[-5]
-4	3	5	[-3,4,-5]
-3	3	3	[-3]
-3	1	5	[-1,2,-3,4,-5]
-2	2	5	[2,-3,4,-5]
-2	1	3	[-1,2,-3]
-1	1	1	[-1]
-1	4	5	[4,-5]
-1	2	3	[2,-3]
1	3	4	[-3,4]
1	1	2	[-1,2]
2	2	2	[2]
2	1	4	[-1,2,-3,4]
3	2	4	[2,-3,4]
4	4	4	[4]

Note that your program should not print all sublists like above but only those in a smallest k set. For $k = 3$, the output from your program should be either

```
size    i    j    sublist
  -5    5    5    [-5]
  -4    3    5    [-3,4,-5]
  -3    3    3    [-3]
```

or

```
size    i    j    sublist
  -5    5    5    [-5]
  -4    3    5    [-3,4,-5]
  -3    1    5    [-1,2,-3,4,-5]
```

(There are two equally good alteratives since there are two sublists with size -3.)

4 About the implementation

You must implement your own (recursive) functions and use them in your program. With the exception of the following, you are *not allowed* to use ready-made functions and operators from `Prelude.hs` or other libraries:

- Those mentioned in Chapter 3 and listed in Section 5.8.
- The higher-order functions `map`, `filter`, and `foldr`.
- `putStr` to print a string.

The reason for this restriction is the this is primarily a course on how to implement from scratch and not a course on how to use what someone else has already provided.

4.1 Haskell programs

In general, Haskell programs are compositions of functions (subprograms) that, in sequence, refine intermediate results into final results. They act like tubes into which arguments are inserted at one end and results pop out at the other end.

Your first step is therefore to identify what functions you need in your program to go from a list (ℓ) and an integer (k) to having a smallest k set printed on the screen. Break down the problem into subproblems, implement functions that solve the subproblems, and use them to solve the original problem. To get you going, here are two passages from the course book that explains this.

- Section 6.4 in the course book, and the exercises that follow until the end of the chapter, make use of this technique of breaking up a problem into subproblems.
- Section 10.8 also contains an example in which a problem is broken up into subproblems but, more importantly, it also shows how types of functions are aligned and how the final program is composed. The operator `>.>` used on page 188 is defined in Section 10.2. It composes functions like the composition operator (`.`) but in the "other direction".

Read these sections for inspiration on how to find and implement your own functions.

4.2 Formatting and printing output

You program will typically be the composition of a function that computes a smallest k set and a function the puts together a string representing the set. The string contains the rows of the output with blanks or tab characters (`'\t'`) to format each line and new line characters (`'\n'`) to break lines.

To print the string, use the function `putStr`. Write, for instance,

```
smallestKset xs k = putStr (yourProgram xs k)
```

assuming `yourProgram` returns a string suitable for printing.

5 Test cases

Test case 1

In this example, $k = 15$. The list used is `[x*(-1)^x | x <- [1..100]]`. (A part of the print-out of the list has been omitted below because of space limitations.) Note that the list used above in the text is the first 5 elements of this list.

```
Entire list: [-1,2,-3,4,-5, <a lot omitted here> , 100]

size    i    j    sublist
 -99   99   99    [-99]
 -98   97   99    [-97,98,-99]
 -97   97   97    [-97]
 -97   95   99    [-95,96,-97,98,-99]
 -96   95   97    [-95,96,-97]
 -96   93   99    [-93,94,-95,96,-97,98,-99]
 -95   95   95    [-95]
 -95   93   97    [-93,94,-95,96,-97]
 -95   91   99    [-91,92,-93,94,-95,96,-97,98,-99]
 -94   93   95    [-93,94,-95]
 -94   91   97    [-91,92,-93,94,-95,96,-97]
 -94   89   99    [-89,90,-91,92,-93,94,-95,96,-97,98,-99]
 -93   93   93    [-93]
 -93   91   95    [-91,92,-93,94,-95]
 -93   89   97    [-89,90,-91,92,-93,94,-95,96,-97]
```

Test case 2

Here, $k = 6$ and the list is `[24,-11,-34,42,-24,7,-19,21]`.

```
Entire list: [24,-11,-34,42,-24,7,-19,21]

size    i    j    sublist
 -45    2    3    [-11,-34]
 -39    2    7    [-11,-34,42,-24,7,-19]
 -36    5    7    [-24,7,-19]
 -34    3    3    [-34]
 -28    3    7    [-34,42,-24,7,-19]
 -27    2    5    [-11,-34,42,-24]
```

Test case 3

Here, $k = 8$ and the list is `[3,2,-4,3,2,-5,-2,2,3,-3,2,-5,6,-2,2,3]`

```
Entire list: [3,2,-4,3,2,-5,-2,2,3,-3,2,-5,6,-2,2,3]

size    i    j    sublist
  -8    6   12    [-5,-2,2,3,-3,2,-5]
  -7    6    7    [-5,-2]
  -7    3   12    [-4,3,2,-5,-2,2,3,-3,2,-5]
  -6   10   12    [-3,2,-5]
  -6    5   12    [2,-5,-2,2,3,-3,2,-5]
  -6    3    7    [-4,3,2,-5,-2]
  -5   12   12    [-5]
  -5    6   10    [-5,-2,2,3,-3]
```

6 How to present your work

At the grading you should first show the results of running your program on each of the test cases in Section 5. The output should look very similar, with aligned columns. Prepare this in advance. Then you should show:

1. How sublists are represented and computed (top-down).
2. The code that does the sorting.
3. The code that prints.

Make sure you can explain and motivate all your code.