Smallest k sets

– Lab assignment 1 –
– D7012E Declarative programming –

Håkan Jonsson

March 28, 2017

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 unique integers, computes and prints a *smallest k set* of ℓ. This is a set of sublists defined using the following.

- Deleting zero or more elements from the ends of a list – not from within – results in a shorter (possibly even empty) *sublist*.
- The *size* $\sigma(\ell)$ of a non-empty list ℓ is the sum of all its elements. Empty lists have no size.

Now, 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,

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

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

2 Details

If we order all sublists after size, so the smallest sublists end up at the beginning, the k first sublists constitute a smallest k set. However, since sizes are not necessarily unique, there could be several ways to order the sublists sizewise and each could give rise to a smallest k set different from the other.

An example is shown in the table below that contains all sublists of `[-1,2,-3,4,-5]` listed top-down from smallest to largest and one sublist per row. In the table, columns `i` and `j` contain start and end indices[1] of sublists. The top 3 sublists is a smallest k set for $k = 3$ but the same is true for the top two together with the 4^{th} sublist. This is because the sums of the sizes of their sublists are equal.

```
Entire list: [-1,2,-3,4,-5]

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. In this case (for $k = 3$), the output from your program should be either

```
size    i    j    sublist
```

[1] Here, indices start with 1.

```
  -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]
```

Which one your program computes and prints does not matter. 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.

Note that there could be negative as well as positive integers in the list. Also, lists in a smallest k set may overlap, as the example shows.

2.1 Implementation

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

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 solves 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.

2.2 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 a smallest k. The string contains the rows of the output (with blanks to format each line so it looks like in the example) separated by the special character `'\n'` that makes the printing continue on the next row.

To print the string, use the function `putStr`.

```
main xs k = putString (yourProgram xs k)
```

assuming `yourProgram` returns a string suitable for printing.

3 Appendix: Functions and operators from Prelude.hs

Please feel free to use the following from `Prelude.hs` in working with the assignment.

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