Department of Mathematics and Computer Science
University of Southern Denmark, Odense

November 26, 2008
KSL

DM509 Exam

Obligatory Assignment, part 2: HASKELL

Kim Skak Larsen

Fall 2008

Introduction

In this note, we describe one part of the exam project which must be solved in connection with the course DM509, Programming Languages, Fall 2008. It is important to read through the entire project description before starting the work on the project; also the sections on requirements and how to turn in your solution.

Deadline

Wednesday, December 17, 2008 at noon.

Seat Reservations

We will investigate the problem that train companies such as DSB have when people buy seat reservations. At the time of purchase, people are told which seat they get (in Denmark, this is done by giving people a car number and a seat number in that given car). It turns out that it is harder to place people in a fashion that utilizes the full train capacity when reservation have to be treated immediately (also called “online”) instead of when all requests for reservations are known (called “offline”).

In this problem, we assume that we have a train which travels through k stations (including the start and end station) numbered from zero. The train has n seats that are also numbered from zero. All reservations are made before the train starts from station zero.

A request for a reservation is a tuple (x, y) representing a trip from station x to y, i.e., $x, y \in \{0, \ldots, k-1\}$ and $x < y$.

You have to treat a list of reservations as for instance

```
[(0,1),(2,3),(0,2),(1,3),(1,2)]
```

You are given a starting point for your solution which can be seen in Fig. 1.

First you must write a function `onlineFirstFit` which implements an online first-fit strategy. This means that, as it is the case for DSB, you must consider the reservation requests in order and process them one at a time. A request for a reservation can be accommodated if there exists a seat which is available for the entire duration of the trip. Otherwise, the reservation is rejected. One cannot place a person on multiple seats for different parts of the trip.

With the input above and $k = 4$ stations and $n = 2$ seats, we get the result in Fig. 2. Here, a number on an interval refers to the index of the reservation request in the input. Stations are listed horizontally and seats vertically. Notice that under the full line, the request with index 3 appears because it could not be accommodated on a seat and was therefore rejected.

```
-- IMPORTS

import List(sort)


-- TYPES

type Seat = [Int]
type Train = [Seat]
type Reservation = (Int,Int)


-- ONLINE FIRST-FIT

onlineFirstFit :: Int -> Int -> [Reservation] -> Train
onlineFirstFit n k requests = ...


-- OFFLINE LEFT-TO-RIGHT FIRST-FIT

offlineLeftToRight :: Int -> Int -> [Reservation] -> Train
offlineLeftToRight n k requests = onlineFirstFit n k (sort requests)


-- STATISTICS

ratio :: Int -> Int -> [Reservation] -> Float
ratio n k requests = ...
```

Figure 1: Starting point for file `seats.hs`.

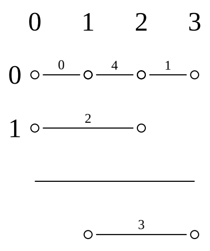

Figure 2: A small example.

The function `onlineFirstFit` must compute a representation of the *accommodated* reservation requests corresponding to this illustration, i.e., in the given example,

```
onlineFirstFit 2 4 [(0,1),(2,3),(0,2),(1,3),(1,2)]
```

must return

```
[[0,4,1],[2,2,-1]]
```

Thus, rejected requests are not listed.

Here -1 represent "unoccupied". Note that reservations longer than one are represented by repeating the index of the reservation. In detail, a reservation (x, y) with index i in the request sequence is represented by the value i in positions $x, x+1, \ldots, y-1$ of the given seat. In the example, the reservation $(0, 2)$ has index 2 in the input sequence, so since it could be accommodated on seat 1, this seat has a 2 in positions 0 and 1 in the list representing seat 1.

Having implemented this, an offline strategy which first sorts the input lexicographically and then uses first-fit can be defined. This is already done in the script. You must now implement the function `ratio` which is defined to be the number of accommodated reservations in the online case divided by the number of accommodated reservations in the offline case.

This ratio says something about how much harder it is to solve the online problem, e.g., a ratio of 3 would say that we do 3 times worse in the online case than it would have been possible to do if we had known all the reservation requests in advance, so that we could treat them offline.

As an example, consider the sequence `[(0,1),(2,3),(0,2),(1,3)]`. Here,

```
onlineFirstFit 2 4 [(0,1),(2,3),(0,2),(1,3)]
```

would accommodate 3 of the requests, rejecting the last.

However, `offlineLeftToRight` would first sort the sequence, producing

```
[(0,1),(0,2),(1,3),(2,3)],
```

and then call

```
onlineFirstFit 2 4 [(0,1),(0,2),(1,3),(2,3)]
```

which would accommodate all of them. Thus, the ratio here would be $4/3$.

Execution Requirements

The programs must be written in HASKELL and must execute correctly using the HUGS system on the department's computers, i.e., the ones in the terminal room. Example computers are

`logon`⟨digit⟩`.imada.sdu.dk`,

where ⟨digit⟩ can be any of the digits $1, \ldots, 9$. These are the computers you can connect to using `ssh` from outside IMADA.

You are very welcome to develop your programs at home, but it is your responsibility. This includes technical problems at home, lack of access to relevant software, moving data to IMADA via modem, e-mail, ftp, floppy disks, USB keys, etc. and converting to the correct format, e.g., between Windows and Linux.

Turning In

One part of what you hand in must be a HASKELL script called `seats.hs` (all lower case). You must equip all functions you define with a type annotation.

You should produce a small report describing what you have done. A sufficient collection of tests must be carried out, and you must verify and document in the report that the correct results are produced. The report must also contain a complete listing of programs.

Possible omissions, known errors, etc. should be described in the report. It is often a good idea to do this in a separate section instead of mixing it in with the rest of the report.

All the material described above must be turned in on paper at the secretaries' office and also electronically via Blackboard.

The paper version and the electronic version must of course be identical, except that you must date and sign the paper version. Also, for the purpose of the exam protocol, please clearly indicate your full name and birthday to make sure that we correctly identify you. It is also a good idea to write your IMADA login as well as your preferred e-mail address.

The secretaries' office may be closed for very short periods of time. If, for some unexpected reason, the office must be closed for longer periods of time close to the deadline, an announcement will be made outside the office, giving instructions as to where you turn in your report.

Exam Rules

An obligatory assignment is an exam project. Cooperation beyond what is explicitly permitted will be considered cheating and will be treated as such. You have a duty to keep your notes private and protect your files against reading and copying by others. Both parties involved in a possible plagiarism can be held responsible.

There will be given what we judge to be more than sufficient time for each assignment and you are strongly encouraged to plan your work such that you will finish some days before the deadline.

Assignments which are turned in after the deadline will not be accepted. Downtime on the system or the printers will not automatically result in an extension; not even if it is the last hours before the deadline. Neither will own or children's illness without a statement from your physician, etc.