Competitive Programming in Haskell: Classrooms

Christian Pletbjerg

July 25, 2023

Contents

1 Introduction

Let's say the (dreaded) club day at your school is approaching, and you've been (unfortunately) tasked with assigning clubs' activities at specified start times and finish times to a limited number of classrooms. Clubs despise sharing so if given a classroom, they must have the classroom completely to themselves from their activity's start time to finish time (inclusive). As there are only a limited number of classrooms, you'd like to maximize the number of activities that can be scheduled.

This problem is exactly the Classrooms problem on Kattis which has a difficulty of 7.0 (Hard)! The solution presented is a (somewhat) tricky greedy algorithm. There aren't very many Haskell solutions, so I figured I'd toss one out here in the mix.

As an overview, we will

- Define notation.
- Design a greedy algorithm (with pseudo code) while proving its correctness.
- Give an implementation in Haskell.

2 Notation

We fix some notation for the rest of the document.

Suppose we have a nonempty set of n activities $A = \{1, \ldots, n\}$, and a nonempty set of m classrooms $C = \{1, \ldots, m\}$. Each activity $i = 1, \ldots, n$ has an associated integer *start time* s_i, and an associated integer *finish time* f_i satisfying $s_i \leq f_i$ and $s_i \geq 1$ i.e., activities must start before they finish, and all activities start at time 1 or later. Often, we will associate an activity i with the closed interval $[s_i, f_i]$ of its finish time and start time. Given two distinct activities i, j, we say that i, j *overlap* (or i *overlaps* with j) if $[s_i, f_i] \cap [s_j, f_j] \neq \emptyset$.

A *schedule* $\mathcal{S}$ is a function

$$\mathcal{S} : A \to C \cup \{\perp\}$$

which *assigns* each activity to a classroom or $\perp$. Note that $\perp$ is not a classroom i.e., $\perp \notin C$. If $\mathcal{S}$ assigns an activity i to a classroom $c \in C$, then we say that i is *scheduled*. A schedule $\mathcal{S}$ is *valid* if all activities in $\mathcal{S}$ that are assigned to the same classroom do not overlap i.e., all distinct activities i, j assigned to the same classroom $c \in C$ (i.e., $\mathcal{S}(i) = \mathcal{S}(j) = c$) satisfy $[s_i, f_i] \cap [s_j, f_j] = \emptyset$.

The *size* of a schedule $\mathcal{S}$ is the number of activities which are scheduled i.e., the size of $\mathcal{S}$ is

$$|\{i \in A : \mathcal{S}(i) \in C\}|. \tag{1}$$

If schedule $\mathcal{O}$ is both valid and of maximum size (w.r.t. valid schedules), then we say that $\mathcal{O}$ is *optimal*. Thus, in the notation given, our problem amounts to finding and optimal schedule.

3 Example

Suppose we are given $n = 5$ activities defined as follows

- $s_1 = 1, f_1 = 4$
- $s_2 = 2, f_2 = 9$
- $s_3 = 4, f_3 = 7$
- $s_4 = 5, f_4 = 8$
- $s_5 = 1, f_5 = 9$

Moreover, suppose we have $m = 2$ classrooms.

One possible schedule $\mathcal{S}$ would be as follows.

$$1 \mapsto 1 \tag{2}$$
$$2 \mapsto 2 \tag{3}$$
$$3 \mapsto \perp \tag{4}$$
$$4 \mapsto 1 \tag{5}$$
$$5 \mapsto \perp \tag{6}$$
$$\tag{7}$$

We can visualize $\mathcal{S}$ as follows.

```
c1: |-----1-----|   |-----4-----|
c2:     |-------------2-------------|
    1   2   3   4   5   6   7   8   9
```

`c1` denotes classroom 1, and `c2` denotes classroom 2. A vertical bar `|` followed by dashes `-`, a number x, more dashes `-`, then another vertical bar `|`, denote activity x's start and finish time (via the vertical bar) where we note that bottom horizontal numbers denote the time.

It's easy to see that this is a valid schedule, and this is in fact an optimal schedule of size 3 for this instance of the problem.

Already, we see an interesting property of this problem. It's clear that an optimal schedule is not unique. With this example here, we instead have $2 \mapsto \perp$ and $5 \mapsto 2$ which is also clearly a valid schedule and also of optimal size 3.

4 An Algorithm

We propose a greedy algorithm to give us an assignment of activities to classrooms of maximum size.

Suppose we are given $m > 0$ classrooms, and $n >$ activities written as follows.

$$[s_1, f_1], \ldots, [s_n, f_n]$$

Without loss of generality, we may assume that the activities satisfy $f_i \leq f_j$ for all $i \leq j$ i.e., the activities are ordered by earliest finish time first.

We will inductively define a schedule $\mathcal{S}$. For every classroom c, we will maintain a variable t_c which will maintain the invariant that t_c is the largest finish time of activities already assigned to classroom c (or 0 if no activities are assigned to classroom c). Then, we define $\mathcal{S}$ as follows. Set $t_c = 0$ for every classroom c. For each activity i (processed from earliest finish time to latest finish time), let c denote the classroom with a largest t_c s.t. $[0, t_c] \cap [s_i, f_i] = \emptyset$. Then, either:

- If a classroom c exists, then assign activity i to the classroom c, and update t_c to f_i (this update is necessary to maintain the invariant the t_c denotes the largest finish time of activities already assigned to classroom c).

- Otherwise, no such classroom c exists, so assign activity i to $\perp$ i.e., activity i will not be scheduled.

Obviously, we have the following properties of $\mathcal{S}$ by construction which can be shown by induction.

Lemma 4.1. *When $\mathcal{S}$ processes activity i and considers assigning it to a classroom, each t_c is the largest finish time of activities assigned to classroom c or 0 if no activities are assigned to classroom c.*

Lemma 4.2. *$\mathcal{S}$ is valid schedule.*

We omit the proof.

Since $\mathcal{S}$ is a valid schedule, to show that $\mathcal{S}$ is an optimal schedule we just need to show that the size of $\mathcal{S}$ is as large as possible w.r.t. valid schedules. This is a bit tricky to show, so we'll first give an overview of the steps. We will:

- Assume that we have an optimal schedule $\mathcal{O}$.
- Show that we can incrementally transform $\mathcal{O}$ into a new optimal schedule $\mathcal{O}'$ that is more similar to $\mathcal{S}$.
- Repeatedly apply the transformation from $\mathcal{O}$ to an optimal schedule $\mathcal{O}'$ until we get the same schedule as $\mathcal{S}$ which shows that $\mathcal{S}$ is optimal as well (since each application of the transformation is also optimal).

Suppose $\mathcal{O}$ is an optimal schedule. Since schedules are functions from activities to classrooms (or $\perp$), we may regard $\mathcal{O}$ as an assignment of the activities ordered by earliest finish time first. Now, let ℓ denote the first activity scheduled such that $\mathcal{O}$ and $\mathcal{S}$ differ. Thus, for every activity $1 \leq i < \ell$, we have that $\mathcal{O}(i) = \mathcal{S}(i)$ We will prove the following theorem whose importance we will see shortly.

Theorem 4.3. *There exists a schedule $\mathcal{O}'$ which is:*

- *optimal;*
- *for every activity j with $1 \leq j \leq \ell$, we have $\mathcal{S}(j) = \mathcal{O}'(j)$ i.e., $\mathcal{O}'$ and $\mathcal{S}$ schedule activity j the same way.*

Proof. We describe how to construct the schedule $\mathcal{O}'$. by distinguishing cases on whether an activitiy is in the range $[1, \ell)$, or is ℓ, or in the range $(\ell, n]$. At each step of the way, we will argue that $\mathcal{O}'$ does not assign an activity to a classroom s.t. the activity will overlap with another already assigned classroom. Finally, we will see that this is optimal as the construction of $\mathcal{O}'$ will be essentially $\mathcal{O}$ which is already known to be optimal.

For every activity $1 \leq i < \ell$, put

$$\mathcal{O}'(i) \mapsto \mathcal{O}(i) = \mathcal{S}(i) \tag{8}$$

i.e., assign each activity i the same way $\mathcal{O}$ does (which we may recall that by defn. of ℓ, this is the same as $\mathcal{S}$). Then, for activity ℓ, put

$$\mathcal{O}'(\ell) \mapsto \mathcal{S}(\ell) \tag{9}$$

i.e., assign activity ℓ the same way $\mathcal{S}$ does.

For the remaining activities $k > \ell$, we must distinguish cases on how $\mathcal{S}$ and $\mathcal{O}$ assign activity ℓ.

- If $\mathcal{S}(\ell) = \perp$ and $\mathcal{O}(\ell) = c \in C$, then this means that $\mathcal{S}$ was unable to assign activity ℓ to a classroom so the largest finish time of every activity assigned to classroom thus far overlaps with activity ℓ. But, since $\mathcal{S}(i) = \mathcal{O}(i)$ for every activity $1 \leq i < \ell$, this would mean that since $\mathcal{O}$ is optimal and hence a valid schedule, we get that there must exist a classroom which does not overlap with activity ℓ – contradicting that $\mathcal{S}$ was unable to assign activity ℓ to a classroom. Hence, this case is impossible.

- Suppose $\mathcal{S}(\ell) = c \in C$ and $\mathcal{O}(\ell) = \perp$. Then, for every activity $k > \ell$, we put

$$\mathcal{O}'(k) \mapsto \begin{cases} \perp & \text{if } \ell, k \text{ overlap and } \mathcal{O}(k) = c \\ \mathcal{O}(k) & \text{otherwise} \end{cases} \tag{10}$$

  Observe that the only activities $\mathcal{O}$ and $\mathcal{O}'$ assign differently are:

  - activity ℓ where we recall $\mathcal{O}'(\ell) = \mathcal{S}(\ell) = c$; and
  - any activity $k > \ell$ for which $\mathcal{O}(k) = c$ and k, ℓ overlap where we put $\mathcal{O}'(k) = \perp$.

  Obviously, this definition makes $\mathcal{O}'$ a valid schedule since $\mathcal{O}$ is a valid schedule, and any activity which could overlap with another activity assigned to the same classroom in $\mathcal{O}'$ must then be assigned to classroom c and overlap with activity ℓ, but such an activity is not scheduled in $\mathcal{O}'$. Thus, $\mathcal{O}'$ is a valid schedule.

  Now, to show that $\mathcal{O}'$ is optimal, all that remains is to show that $\mathcal{O}'$ has the largest size w.r.t. valid schedules. We show this by showing that $\mathcal{O}'$ has the same size as $\mathcal{O}$. Note that $\mathcal{O}'$ assigns ℓ to a classroom, but $\mathcal{O}$ does not assign ℓ to a classroom. So, to show that $\mathcal{O}'$ is optimal, since $\mathcal{O}'$ is essentially identical to $\mathcal{O}$, it suffices to show that there is at most one activity $k' > \ell$ which satisfies ℓ, k' overlap, and $\mathcal{O}(k') = c$. Suppose for the sake of contradiction that $k', k'' > \ell$ are distinct, both overlaps with ℓ, and $\mathcal{O}(k') = \mathcal{O}(k'') = c$. Without loss of generality, we may assume that $k' < k''$. Using that we assumed all activities are ordered by earliest finish time first, we know that

$$f_\ell \leq f_{k'} \leq f_{k''} \tag{11}$$

  So, using that we assumed that both k', k'' overlap with ℓ, it's easy to see that $f_\ell \in [s_\ell, f_\ell] \cap [s_{k'}, f_{k'}]$ and $f_\ell \in [s_\ell, f_\ell] \cap [s_{k''}, f_{k''}]$. Thus, $f_\ell \in$

$[s_{k'}, f_{k'}] \cap [s_{k''}, f_{k''}]$ follows i.e., k', k'' overlap and are assigned to the same classroom c in $\mathcal{O}$ – contradicting that $\mathcal{O}$ is valid.

- Suppose $\mathcal{S}(\ell) = c \in C$ and $\mathcal{O}(\ell) = c' \in C$ where we necessarily have $c \neq c'$ since ℓ is the first activity for which $\mathcal{S}$ and $\mathcal{O}$ differ.

  Then, for every activity $k > i$, we put

$$\mathcal{O}'(k) \mapsto \begin{cases} c' & \text{if } \mathcal{O}(k) = c \\ c & \text{if } \mathcal{O}(k) = c' \\ \mathcal{O}(k) & \text{otherwise} \end{cases} \tag{12}$$

  Observe that the only activities $\mathcal{O}$ and $\mathcal{O}'$ assign differently are:

  – activity ℓ where we recall $\mathcal{O}'(\ell) = \mathcal{S}(\ell) = c \neq c' = \mathcal{O}(\ell)$; and
  – any activity $k > \ell$ for which either $\mathcal{O}(k) = c$ (so $\mathcal{O}'(k) = c'$) or $\mathcal{O}(k) = c'$ (so $\mathcal{O}'(k) = c$).

  It's clear that $\mathcal{O}'$ has the same size as $\mathcal{O}$, and hence has the largest size w.r.t. valid schedules. So, to show that $\mathcal{O}'$ is optimal, we need to show that $\mathcal{O}'$ is valid i.e., all activities assigned to the same classrooms do not overlap. Since $\mathcal{O}$ and $\mathcal{O}'$ are essentially identical, we only need to show that the activities assigned to classrooms c (and c' resp.) do not overlap.

  We first show that no activities assigned to classroom c in $\mathcal{O}'$ overlap. Note that ℓ since $\mathcal{O}'(\ell) = \mathcal{S} = c$, by defn. of $\mathcal{S}$, we know that ℓ does not overlap with any activity $1 \leq i < \ell$ assigned to c in $\mathcal{O}'$. Recall that $\mathcal{O}(\ell) = c'$, so for any activity $k > \ell$ satisfying $\mathcal{O}'(k) = c$, we know that $\mathcal{O}(k) = c'$, so since $\mathcal{O}$ is valid, this implies that no activity k can overlap with any activity previously assigned to c in $\mathcal{O}'$.

  Now, we show that no activities assigned to classroom c' in $\mathcal{O}'$ overlap. Obviously, all activities $1 \leq i < \ell$ assigned to classroom c' in $\mathcal{O}'$ do not overlap.

  Let $t < \ell$ denote the largest activity satisfying $\mathcal{O}'(t) = c$, and let $t' < \ell$ denote the largest activity satisfying $\mathcal{O}'(t') = c'$. Note that $\ell \geq \ell$ is the smallest activity satisfying $\mathcal{O}'(\ell) = c$, and let $\ell' > \ell$ be the smallest activity satisfying $\mathcal{O}'(\ell') = c'$ (so $\mathcal{O}(\ell') = c$). If such activities do not exist, then the following results are trivial.

  We show that t', ℓ' do not overlap. By defn. of $\mathcal{S}$, since $\mathcal{S}(\ell) = c$, we get that

$$f_{t'} \leq f_t \tag{13}$$

  since t' and ℓ do not overlap (since $\mathcal{O}(t') = c'$, $\mathcal{O}(\ell) = c'$, and $\mathcal{O}$ is valid) and $\mathcal{S}$ assigned activity ℓ to the classroom with the largest f_t s.t. ℓ, t do not overlap.

  Then, since we know $\mathcal{O}(t) = c$, $\mathcal{O}(\ell') = c$, and $\mathcal{O}$ is valid, we get that t and ℓ' do not overlap, so since $f_{t'} \leq f_t$, it certainly follows that t' and ℓ' do not overlap as required.

Finally, for any $k > \ell' > \ell$ satisfying $\mathcal{O}'(k) = c'$, we know that $\mathcal{O}(k) = c$, so since $\mathcal{O}$ is valid, this implies that no activity k can overlap with any activity previously assigned to c' in $\mathcal{O}'$.

This concludes all possible cases, and completes the proof of the theorem. □

Finally, we may apply this theorem at most n times (since there are at most n activities in any schedule) to make $\mathcal{O}$ assign activities the same way $\mathcal{S}$ while still being optimal. This immediately proves correctness of our algorithm and is summed up with the following theorem.

Theorem 4.4. *$\mathcal{S}$ is an optimal solution.*

5 An Implementation in Haskell

If we made it through that rather wordy proof, the implementation in Haskell shouldn't be that tricky. We need to do two things:

1. Sort the input activities by earliest finish time.
2. Maintain an efficient way find the classroom with the largest finish time of all already assigned activities that does not overlap with the next activity's start time.

We know that we can sort lists in $O(n \log n)$ time. Moreover, we can use an ordered tree of size m to represent classrooms to efficiently find the largest free time which does not overlap with the next activity's start time in $O(\log m)$. We must repeat this operation n times, so this will have $O(n \log m)$ complexity. Altogether, this will have complexity $O(n \log n + n \log m)$. A wrinkle when implementing this is that sometimes classrooms will have the same free time, so be careful to not use a set – rather we need a multiset.

Luckily, there are some libraries on the Kattis server which proides us with these mechanisms. In particular, `Data.List.sortOn` can sort a list in $O(n \log n)$, and we may use an `Data.IntMap.Strict.IntMap` to implement a multiset that can be used to query the largest free time which does not overlap with the next activity's start time in $O(\log m)$ with the function `Data.IntMap.Strict.lookupLT`.

Concretely, an implementation is as follows.

```
import qualified Data.Tuple as Tuple
import qualified Data.List as List
import qualified Data.Maybe as Maybe

import Data.ByteString.Char8 (ByteString)
import qualified Data.ByteString.Char8 as B

import Data.IntMap.Strict (IntMap)
import qualified Data.IntMap.Strict as IntMap
```

```
unsafeReadInt :: ByteString -> Int
unsafeReadInt = fst . Maybe.fromJust . B.readInt

main :: IO ()
main = B.interact
    ( format
    . runner
    . parse
    )

-- | 'parse' parses the given kattis input s.t. we have
-- > (n, m, list of activities' start and finish time)
parse :: ByteString -> (Int, Int, [(Int,Int)])
parse
    = go
    . map B.words  -- [ByteString] -> [[ByteString]]
    . B.lines      -- ByteString -> [ByteString]
  where
    go :: [[ByteString]] -> (Int, Int, [(Int,Int)])
    go ([n,m]:activities) =
        ( unsafeReadInt n
        , unsafeReadInt m
        , map (\[s, f] -> (unsafeReadInt s, unsafeReadInt f)) activities
        )
    go _ = error "Invalid input"

-- | 'format' formats the resulting answer into the desired kattis output
format :: Int -> ByteString
format = B.pack . show

-- | 'runner' executes the main logic of this algorithm.
--
--      1. Sorts according to the heuristic
--
--      2. Schedules an activity according to the classroom with the latest
--      compatible start time
runner :: (Int, Int, [(Int,Int)]) -> Int
runner (n,m,activities)
    = fst
    List.foldl' go (0, initClassrooms) List.sortOn snd activities -- sort according to
  where
    -- initially all @m@ classrooms have t_c of 0.
    -- the IntMap maps free time to its multiplicity.
    initClassrooms :: IntMap Int
    initClassrooms = IntMap.singleton 0 m
```

```
go :: (Int, IntMap Int) -> (Int, Int) -> (Int, IntMap Int)
go (ans, classrooms) (s,f) =
    case IntMap.lookupLT s classrooms of
        Just (classroom, _multiplicity) ->
            -- we schedule the activity
            ( 1 + ans
            , insertClassroom f (deleteClassroom classroom classrooms)
            )
        Nothing ->
            -- or we don't schedule it
            (ans, classrooms)

-- deleting an element in the multiset
deleteClassroom :: Int -> IntMap Int -> IntMap Int
deleteClassroom =  IntMap.alter go
  where
    go Nothing = Nothing
    go (Just count)
        | count == 1 = Nothing
        | otherwise = Just (count - 1)

-- inserting an element in the multiset
insertClassroom :: Int -> IntMap Int -> IntMap Int
insertClassroom = IntMap.alter go
  where
    go Nothing = Just 1
    go (Just count) = Just (count + 1)
```