Recitation 8

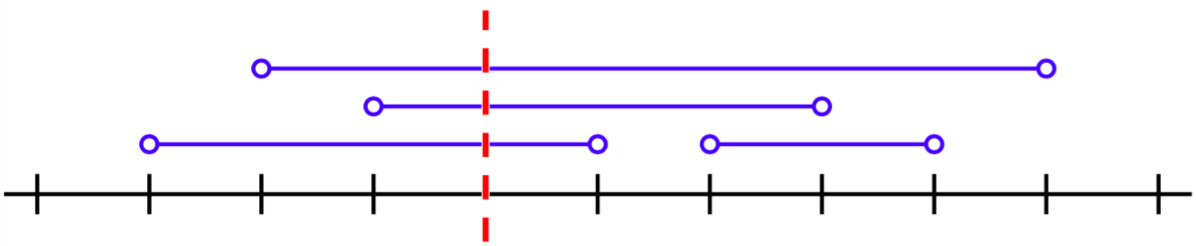
Augmented Tables

8.1 Announcements

- *RangeLab* has been released, and is due *Friday afternoon*.
- *BridgeLab* will be released on Friday. It's not due for two weeks, so enjoy your spring break!

8.2 Interval Checking

Suppose you're given a set of intervals $I \subset \mathbb{Z} \times \mathbb{Z}$ and some $k \in \mathbb{Z}$, and you're interested in determining whether or not there exists $(l, r) \in I$ such that $l < k < r$. For simplicity, let's assume that no two intervals share an endpoint.



Task 8.1. *Implement a function*

val $\mathtt{intervalCheck} : (\mathtt{int} * \mathtt{int})\ \mathtt{Seq.t} \to \mathtt{int} \to \mathtt{bool}$

where $(\mathtt{intervalCheck}\ I\ k)$ *answers the query mentioned above. Your function must be staged such that the line*

val $q = \mathtt{intervalCheck}\ I$

performs $O(|I| \log |I|)$ *work and* $O(\log^2 |I|)$ *span, while each subsequent call* $q(k)$ *only performs* $O(\log |I|)$ *work and span. Try solving this problem with augmented tables.*

We'll store each (l, r) in a table as $(l \mapsto r)$, and augment the table with the function *max*. This allows us to determine the rightmost endpoint of a set of intervals in constant time. To answer the query, we can split I at k to get a set I' of all intervals which begin before k. We then just need to check if any of these have endpoints which are greater than k.

Algorithm 8.2. *Interval Checking with Augmented Tables.*

```
structure Val =
struct
  type t = int
  val f = Int.max
  val I = −∞
  val toString = Int.toString
end

structure Table = MkTreapAugTable (structure Key = IntElt
                                   structure Val = Val)

fun intervalCheck I =
  let
    val T = Table.fromSeq I
    fun query k =
      let val (T′,_,_) = Table.split (T,k)
      in (|T′| > 0) ∧ (Table.reduceVal T′ > k)
      end
  in
    query
  end
```

8.3 Interval Counting

Now suppose you want to solve a more general problem. Given I and k, you want to return $|\{(l,r) \in I \mid l < k < r\}|$. Once again, for simplicity, we'll assume all endpoints are distinct.

Task 8.3. *Implement a function*

```
val intervalCount : (int * int) Seq.t → int → int
```

where (`intervalCheck I k`) *answers the interval counting query as mentioned above. Your function must be staged, just like Task 8.1.*

Similar to parentheses matching, we can use a counter which "increments" at the beginning of each interval, and "decrements" at the end. This corresponds to building a table of $(l \mapsto 1)$ and $(r \mapsto -1)$ for each interval (l,r), and augmenting the table with addition. After splitting this table at k, we can determine the number of "unmatched" intervals on the left in $O(1)$ time.

We have to be careful about off-by-one errors, though: if an interval ends at k, we need to subtract 1. This is handled on line 19 below.

Algorithm 8.4. *Interval Counting with Augmented Tables.*

```
structure Val =
struct
  type t = int
  val f = op+
  val I = 0
  val toString = Int.toString
end

structure Table = MkTreapAugTable (structure Key = IntElt
                                   structure Val = Val)

fun intervalCount I =
  let
    val L = Seq.map (fn (l,_) => (l,1)) I
    val R = Seq.map (fn (_,r) => (r,~1)) I
    val T = Table.fromSeq (Seq.append (L,R))
    fun query k =
      let val (T',co,_) = Table.split (T,k)
          val c = case co of SOME ~1 => ~1 | _ => 0
      in Table.reduceVal T' + c
      end
  in
    query
  end
```