COMP 212 Spring 2022
Homework 09

This homework will be a chance to get used to using modules, which will be used heavily in the final assignment next week.

To compile the code for this homework, do

```
- CM.make "sources.cm";
```

1 Mergable Dictionaries

In lab and class, we discussed the implementation of the `TreeDict` module, which implements dictionaries as trees. In this problem, you will extend the implementation of dictionaries to the following signature:

```
signature DICT =
sig
  type ('k,'v) dict

  val empty   : ('k,'v) dict
  val insert  : ('k * 'k -> order) * ('k,'v) dict  * ('k * 'v) -> ('k,'v) dict
  val lookup  : ('k * 'k -> order) * ('k,'v) dict  * 'k -> 'v option

  val size    : ('k,'v) dict  -> int

  (* computes the sequence of all (key,value) pairs in the dictionary,
     ordered from smallest key to largest key
     *)
  val toSeq : ('k,'v) dict  -> ('k * 'v) Seq.seq

  (* merge (cmp, combine, d1,d2) == d where
     - k in d if and only if k is in d1 or k is in d2
     - If k~v in d1 and k is not in d2, then k ~ v in d
     - If k~v in d2 and k is not in d1, then k ~ v in d
     - If k~v1 in d1 and k~v2 in d2, then k ~ combine (v1, v2) in d
     *)
  val merge  : ('k * 'k -> order) * ('v * 'v -> 'v)
```

```
             * ('k,'v) dict  * ('k,'v) dict
             -> ('k,'v) dict

end
```

Your job is to implement toSeq and merge.

For merge, the idea is to combine two dictionaries into one. We say that a key is in a dictionary if there is some key in the dictionary that is EQUAL to it using the provided comparison function. If a key is in one dictionary with a value, but is not in the other dictionary, it should be in the restul with that value. If a key is in both dictionaries, its values should be combined with the provided function. This merge should follow the outline given by the splitAt and merge functions from Lecture 10, though you will need to update that code to manipulate the values stored in the tree. In place of splitAt, I recommend writing a helper function that is given a tree and a key k and returns three things: the tree of everything less than k, the tree of everything greater than k, and the value stored with k if k was in the tree (using an option).

Task 1.1 (40 pts). Implement toSeq (10 points), splitAt (15 points), and merge (15 points).

Note: you can uncomment the module TestDict and then do TestDict.test() to run some tests.

2 Client Code

In the next homework, you will be doing some data analysis and machine learning. To warm up for that, you will write some client code that uses dictionaries to gather some simple statistics. For this problem, a *document* is a sequence strings, where each string represents a word in the document, and we define the type documents to be a sequence of documents.

```
signature STATISTICS =
sig

    type documents = (string Seq.seq) Seq.seq

    (* given a collection of documents, compute a dictionary
       mapping each word to the number of times it occured *)
    val frequencies : documents -> (string, int) Dict.dict

    (* given a collection of documents, compute the total number
       of distinct words in the documents,
       i.e. the number of different words that occured, counted once each.
    *)
    val num_distinct_words : documents -> int
    (* given a collection of documents, compute a sequence (without duplicates)
       of all of the words in the documents *)
    val distinct_words : documents -> string Seq.seq

    (* given a collection of documents, compute the total number of words
       (counting duplicates more than once) in the documents *)
    val num_words : documents -> int

end
```

For example, if we have two documents

```
this is document one
this is document two
```

this will be represented by a sequence of sequences like so

```
< <"this", "is", "document", "one">,
  <"this", "is", "document", "two">>
```

For this example

- The frequencies are

  ```
  "document" 2
  "is" 2
  "one" 1
  "this" 2
  "two" 1
  ```

- The number of distinct words is 5.
- The sequence of distinct words is `<"document","is","one","this","two">`
- The number of words is 8.

Your job is to implement a module `Stats : STATISTICS` as a client of the `Dict` structure. For full credit, `frequencies` must have sublinear (in the number of documents) span.

Hint: though there are many ways to do some of these functions, if you compute the frequency dictionary first, everything else can be computed from that. Also, in addition to the sequence operations we have used a lot so far, you might find `Seq.flatten : 'a Seq.seq Seq.seq -> 'a Seq.seq` (like the flatten function from Homework 7) to be helpful for this problem.

Task 2.1 (35 pts). Implement `freq` (15 points), `distinct_words` (5 points), `num_distinct_words` (5 points), `num_words` (10 points).

You can uncomment the module `TestStats` and then do `TestStats.test()` to run some tests.

3 NON-COLLABORATIVE CHALLENGE PROBLEM: Testing

Remember that non-collaborative challenge problems are to be done independently. You are not allowed to communicate with anyone about the problems, except to ask the instructor or TAs clarification questions (not hints). Additionally, you are not allowed to search for help on the specific problem from any sources besides the course materials.

See the problem description in `hw09challenge.sml`.