COMP 3958: Lab 4

Submit a zip file containing the `kvtree` directory & the file `primes.ml`. The `kvtree` directory should contain all source files for question 1. Do not need submit `_build` directories. Note that if your program does not build, you may receive no credit for it. Maximum score: 13

1. In class, we have seen how to generalize our original implementation of binary search trees to use a comparison function. We further improve it by creating a functor that takes a module containing a comparison function and returns a module of binary search trees. In the last lab, you implemented a binary search tree of key-value pairs. It is clear that a comparison function can be specified to compare keys. For this exercise, you are asked to implement a module named `Kvtree` that has a functor named `Make` that creates modules of binary search trees of key-value pairs. `Make` takes a module of type `OrderedType` defined as follows:

   ```
   module type OrderedType = sig
     type t
     val compare : t -> t -> int
   end
   ```

   To facilitate testing, do not make the tree type abtract.

   Besides the 4 functions specified in lab 3 (which should be renamed to `insert`, `find`, `delete` & `of_list`), you need to provide 3 additional functions:

   (a) `empty` that returns an empty tree

   (b) `is_empty` that tests whether a tree is empty

   (c) `to_list` that returns a list of key-value pairs that represents the tree; it is the inverse operation of `of_list`, i.e., applying `to_list` followed by `of_list` to a tree returns the original tree; applying `of_list` followed by `to_list` to a list returns the original list

   When one of these functions take a tree as an argument, make the tree the last argument.

   Create a directory named `kvtree` and put your implementations in that directory. There should be a file named `kvtree.ml` and its corresponding interface file `kvtree.mli`. You may use additional files if necessary. Make sure your system can be built using the command: `ocamlbuild kvtree.cmo`

2. Consider the following set of 6-digit primes:

   $$788999, 889997, 897899, 979889, 988979, 997889, 998897.$$

   It is easy to see that the numbers are permutations of one another. This set consists of 7 elements.

   Similarly, the following is a set of eleven 6-digit primes that are permutations of one another.

   $$788789, 788897, 798887, 878789, 878987, 887987, 888779, 889877, 897887, 898787, 988877.$$

   For this part of the lab, you are asked to implement a stand-alone program that uses the OCaml `Map` module (with the `Make` functor) in the standard library to find the size of the largest set of 6-digit primes that are permutations of one another.

   A list of 6-digits primes is provided in the file `6-digit-primes.txt`. Your program needs to read them in (via `stdin` using I/O redirection) & process them. The program prints out the size of the largest set of those primes that are permutations of one another.

   Name your file `primes.ml` & make sure it can be built using the command: `ocamlbuild primes.native`