Sergio E. Garcia Tapia
Algorithms by Sedgewick and Wayne (4th edition) [SW11]
January 07, 2025

3.5: Applications

Exercise 1. Implement `SET` and `HashSET` as "wrapper class" clients of `ST` and `HashST`, respectively (provide dummy values and ignore them).

Solution. See `com.segarciat.algs4.ch3.sec5.ex01`.

Exercise 2. Develop a `SET` implementation by starting with the code for `SequentialSearchST` and eliminating all the code involving values.

Solution. See `com.segarciat.algs4.ch3.sec5.ex02`.

Exercise 3. Develop a `SET` implementation `BinarySearchSET` by starting with the code for `BinarySearchST` and eliminating all the code involving values.

Solution. See `com.segarciat.algs4.ch3.sec5.ex03`.

Exercise 4. Develop classes `HashSTint` and `HashSTdouble` for maintaining sets of keys of primitive `int` and `double` types, respectively. (Convert generics to primitive types in the code of `LinearProbingHashST`).

Solution. See `com.segarciat.algs4.ch3.sec5.ex04`.

Exercise 8. Modify `LinearProbingHashST` to keep duplicate keys in the table. Return *any* value associated with the given key for `get()`, and remove *all* items in the table that have keys equal to the given key for `delete()`.

Solution. See `com.segarciat.algs4.ch3.sec5.ex08`.

Exercise 9. Modify `BST` to keep duplicate keys in the tree. Return *any* value associated with the given key for `get()`, and remove *all* items in the table that have keys equal to the given key for `delete()`.

Solution. See `com.segarciat.algs4.ch3.sec5.ex09`.

Exercise 12. Modify `LookupCSV` to associate with each key all values that appear in key-value pairs with that key in the input (not just the most recent, as in the associative-array abstraction).

Solution. See `com.segarciat.algs4.ch3.sec5.ex12`.

Exercise 13. Modify `LookupCSV` to make a program `RangeLookupCSV` that takes two key values from the standard input and prints all key-value pairs in the `.csv` file such that the key falls within the range specified.

Solution. See `com.segarciat.algs4.ch3.sec5.ex13`.

Exercise 14. Develop and test a static method `invert()` that takes as argument an `ST<String, Bag<String>>` and produces as return value the inverse of the given symbol table (a symbol table of the same type).

Solution. See `com.segarciat.algs4.ch3.sec5.ex14`.

Exercise 15. Write a program that takes a string on standard input and an integer k as command-line argument and puts on standard output a sorted list of the k-grams (substrings of length k) found in the string, each followed by its index in the string.

Solution. See `com.segarciat.algs4.ch3.sec5.ex15`.

Exercise 16. Add a method `sum()` to `SparseVector` that takes a `SparseVector` as argument and returns a `SparseVector` that is the term-by-term sum of this vector and the argument vector. *Note*: You need `delete()` (and special attention to precision) to handle the case when an entry becomes 0.

Solution. See `com.segarciat.algs4.ch3.sec5.ex16`.

Exercise 17. *Finite mathematical sets.* Your goal is to develop an implementation of the following API for processing finite mathematical sets:

```
     public class MathSET<Key>
MathSet(Key[] universe)     // Create the empty set (using given universe)
void add(Key key)                // put key into the set
MathSET<Key> complement()               // set of keys int he universe that
   are not in this set.
void union(MathSET<Key> a)       // put any keys from a into the set that are
   not
// already there.
void intersection(MathSET<Key> a) // remove any keys from this set that are
   not in a.
void delete(Key key)             // remove key from the set
boolean contains(Key key)          // is key in the set?
boolean isEmpty()                  // is the set empty?
int size()                       // number of keys in the set
```

Solution. See `com.segarciat.algs4.ch3.sec5.ex17`.

References

[SW11] Robert Sedgewick and Kevin Wayne. *Algorithms*. 4th ed. Addison-Wesley, 2011. ISBN: 9780321573513.