Exercises 9 (Set ADTs)

9A (This exercise is drill for anyone unfamiliar with sets.)

(a) Using {…} notation, write down:

(i) the set of primary colors (call this set *primary*);

(ii) the set of colors of the rainbow (*rainbow)*;

(iii) the set of colors in your country's national flag (*flag*).

What is the size of each set?

(b) In terms of the above sets, and the set operations $\supseteq$, $\cup$, $\cap$, and $-$, write down expressions representing:

(i) the set of rainbow colors in your national flag;

(ii) the set of rainbow colors not in your national flag;

(iii) the proposition that all colors in your national flag occur in the rainbow;

(iv) the proposition that your national flag contains exactly the primary colors.

What is the value of each expression?

9B Consider Eratosthenes' sieve algorithm in the course notes. Show that removing step 2.1 (but keeping step 2.1.1) would make the algorithm less efficient, but still correct.

9D Suppose that we wish to design a contract for immutable sets. In the `Set` interface, the mutative transformer `addAll` is to be replaced by an applicative transformer that returns the union of this set and a second set (without updating either set). The other mutative transformers are to be replaced likewise.

(a) Modify the `Set` interface accordingly.

(b) Outline how the array implementation of sets would be affected.

9E Modify the array representation of bounded sets, such that the set members are left unsorted.

(a) Draw diagrams showing *at least two* possible representations of the set of words in the following line, assuming that *cap* = 6:

to be, or not to be

(b) How would the array implementation be affected?

(c) Summarize the algorithms and their complexities. Comment on your results.

9F Modify the array representation of bounded sets, such that the size is not stored explicitly. What are the consequences?

9G Suggest suitable representations for:

(a) sets of ISO-LATIN-1 (8-bit) characters;

(b) sets of Unicode (16-bit) characters.

Given suitable sets of ISO-LATIN-1 characters, how would you test whether a given character is a digit? a letter? a letter or digit?

9H Suggest a suitable representation for sets of countries.

* **9J** A ***bag*** (or *multiset*) is similar to a set, but it may contain several instances of the same member. For example, {'to', 'be', 'or', 'not', 'to', 'be'} is a bag of words, which is equal to {'be', 'be', 'not', 'or', 'to', 'to'} (since

order of members is not significant), but is unequal to {‘be’, ‘not’, ‘or’, ‘to’} (since the number of instances of each member is significant). Adding a member always increases the number of instances in the bag by one.

(a) Design a bag ADT. Provide set-like operations, including bag union and bag subsumption (but not bag intersection or difference). In addition, provide an operation that returns the number of instances of a given member.

(b) How would you represent a bag without actually storing multiple instances of the same member?

(c) Show how to implement your bag ADT using a sorted array representation.

(d) Show how to implement your bag ADT using a sorted SLL representation.

Exercises 10 (The BST Data Structure)

10A Consider a BST whose elements are abbreviated names of chemical elements.

(a) Starting with an empty BST, show the effect of successively adding the following elements: H, C, N, O, Al, Si, Fe, Na, P, S, Ni, Ca.

(b) Show the effect of successively deleting Si, N, O from the resulting BST.

10B The following is a recursive formulation of the BST search algorithm.

To find which if any node of the subtree whose topmost node is *top* contains an element equal to *target*:

1. If *top* is null:
   1.1. Terminate with answer *none*.
2. If *top* is not null:
   2.1. If *target* is equal to *top*'s element:
      2.1.1. Terminate with answer *top*.
   2.2. Else, if *target* is less than *top*'s element:
      2.2.1. Find which if any node of the subtree whose topmost node is *top*'s left child contains an element equal to *target*.
      2.2.2. Terminate with that node as answer.
   2.3. Else, if *target* is greater than *top*'s element:
      2.3.1. Find which if any node of the subtree whose topmost node is *top*'s right child contains an element equal to *target*.
      2.3.2. Terminate with that node as answer.

Implement this as a Java method as follows:

```
public static BST.Node search
            (BST.Node top,
             Comparable target);
// Find which if any node of the subtree whose topmost
// node is top contains an element equal to target.
// Return a link to that node (or null if there is none).
```

10C The following is a recursive formulation of the BST insertion algorithm. Note that its answer is a link to the modified subtree.

To insert the element *elem* in the subtree whose topmost node is *top*:

1. If *top* is null:
   1.1. Make *ins* a link to a newly-created node with element *elem* and no children.
   1.2. Terminate with answer *ins*.
2. If *top* is not null:
   2.1. If *elem* is equal to *top*'s element:
      2.1.1. Terminate with answer *top*.
   2.2. Else, if *elem* is less than *top*'s element:
      2.2.1. Insert *elem* in the subtree whose topmost node is *top*'s left child.
      2.2.2. Terminate with answer *top*.
   2.3. Else, if *elem* is greater than node *top*'s element:
      2.3.1. Insert *elem* in the subtree whose topmost node is *top*'s right child.
      2.3.2. Terminate with answer *top*.

Implement this algorithm as a Java method as follows:

```
public static BST.Node insert
            (BST.Node top,
             Comparable elem);
// Insert the element elem in the subtree whose topmost
// node is top. Return a link to the modified subtree.
```

*** 10D** Consider a recursive formulation of the BST deletion algorithm.

(a) Devise a recursive algorithm that deletes the element *elem* in the subtree whose topmost node is *top*. Its answer should be a link to the remaining subtree.

(b) Implement your algorithm in Java.

10E Write Java methods to:

(a) return the depth of a given BST;

(b) return the *k*'th element from the left in a given BST.

10F Verify that pre-order traversal of a BST, followed by inserting the elements one by one into an initially empty BST, will reproduce the original BST exactly.

10G What happens if you replace pre-order traversal by *in-order* traversal in Exercise 10F?

10H Write a Java method that tests whether a given BST is well-balanced.

*** 10J** Write a Java method that balances a given BST if necessary.

Exercises 11 (Map ADTs)

11A Draw diagrams showing how the example maps *Roman*, *NAFTA*, and *EU* from the course notes would be represented by entry arrays, assuming that *cap* = 10.

11B Repeat Exercise 11A using the linked-list representation of maps.

11C Repeat Exercise 11A using the BST representation of unbounded maps.

11D Suppose that we wish to design a contract for immutable maps. In the `Map` interface, the mutative transformers `remove`, `put`, and `putAll` operations are to be replaced by applicative transformers, each of which is to return a new map (without overwriting any map).

(a) Modify the `Map` interface accordingly.

(b) Show how to modify the entry-array implementation accordingly.

(c) Show how to modify the linked-list implementation accordingly.

(d) Show how to modify the BST implementation accordingly.

11E The implementation of the interface `java.util.Set` given by the Java class `java.util.TreeSet` uses a map to represent a set.

(a) How would the set operations be implemented in terms of the map operations in the interface `java.util.Map`?

(b) What are the advantages and disadvantages of this decision?

11F Suppose that the following operations were added to the `Map` interface. The operation `m.containsKey(k)` returns true if and only if map `m` contains an entry with key `k`, and the operation `m.containsValue(v)` returns true if and only if map `m` contains an entry with value `v`. (See the `java.util.Map` interface for a detailed description of these operations).

(a) Show how to modify the entry-array implementation accordingly.

(b) Show how to modify the linked-list implementation accordingly.

(c) Show how to modify the BST implementation accordingly.

11G Consider the following operation: `m.subMap(k1, k2)` returns a new map that contains all of the entries in map `m` whose keys are not less than `k1` and not greater than `k2`.

(a) Show how to modify the entry-array implementation accordingly.

(b) Show how to modify the linked-list implementation accordingly.

(c) Show how to modify the BST implementation accordingly.

11H A ***multimap*** is similar to a map, except there may be several entries with the same key.

(a) How would you represent a multimap without storing multiple copies of the same key?

(b) Show how to implement a multimap ADT using a sorted entry-array representation.

(c) Show how to implement a multimap ADT using a sorted SLL representation.

(d) Show how to implement a multimap ADT using a BST representation.

*** 11J** Design an algorithm that takes an input text file and generates a report listing, in lexicographic order, all words that appear in the file. Each word must be followed by the line numbers where it appears in the file.

For example, given the input file:

```
To be, or not to be,
That is the question.
Whether it is nobler
To die, or to live
And bear the slings and arrows
Of outrageous misfortune.
```

The first few lines of the output should be:

```
and: 5
arrows: 5
be: 1
bear: 5
die: 4
is: 2, 3
it: 3
…
```

Implement your algorithm as a Java program.

Exercises 12 (Hash-Table Data Structures)

12A Consider a hypothetical university in which each student-number is a string of 6 decimal digits. The first two digits indicate the year when the student first registered; the other four digits are a serial number.

The student records are to be stored in a hash-table, in which the keys are student-numbers. Assume the following design for the hash-table:

$m = 100$; $hash(k)$ = first two digits of k

(a) Starting with an empty hash-table, show the effect of successively adding the following student-numbers: 080001, 070001, 070002, 050001, 080002, 050002, 050003.

(b) What is the average number of comparisons when the resulting hash-table is searched for each of these student-numbers?

(c) Show the effect of deleting 080001 from the hash-table.

12B Consider a flight timetable whose keys are flight codes. A flight code consists of an airline code (2 letters) and a serial number (3 or 4 digits).

Design a hash-table to suit this application. Assume that the number of entries is not expected to exceed about 200.

12C Consider a hash-table in which the keys are the names of web servers (such as 'www.amazon.com' and 'www.glasgow.ac.uk').

(a) Suppose that the hash function uses only the first six characters of the web server name. Why is this a bad choice?

(b) Suggest a more suitable hash function.

12D Given a known set of keys, a ***perfect hash function*** is one that translates every key to a different bucket index. For example, given the set of keys {'alpha', 'beta', 'gamma', 'delta'}, the following is a perfect hash function:

$m = 4$; $hash(k)$ = (initial letter of k – 'a') modulo 4

(a) Under what circumstances does it make sense to seek a perfect hash function?

(b) How can a perfect hash function be exploited in implementing a hash-table?

(c) Given the following set of keys:

{CA, MX, US}

find a perfect hash function with $m = 3$.

(d) Given the following set of keys:

{AT, BE, DE, DK, ES, FI, FR,
GR, IE, IT, LU, NL, PT, SE, UK}

find a perfect hash function with $m < 20$.