Assignment 10

Comparing Trees and Hash Tables

README Due: 9:00 PM, Tuesday, April 30, 2002 (*via* `cs70submit`)

Code Due: 9:00 PM, Thursday, May 2, 2002

1 Scenario

Two popular ways to provide maps are trees and hash tables. In the last assignment, you examined hash tables, but you are left wondering whether binary search trees would be a more elegant solution—after all, you don't have to waste time rehashing with binary search trees. You also wonder whether your implementations of trees and hash tables are as efficient as those provided by the STL (and in the case of hash tables, SGI's STL extensions). You don't know yet, but you are determined to find out...

2 High Level Interface and Behavior

In this assignment, you will use the `myspell` program that you wrote in the last assignment as the basis for developing a benchmark to test tree and hash-table implementations. Your program will accept the same arguments as it did in the previous assignment and behave in almost exactly the same way. In addition to the `-d` option described in the last assignment, the program will also accept the following additional options to control the data type that is used to represent all dictionaries:

`-h` Use *HashSet<string>*

`-t` Use *TreeSet<string>*

`-T` Use *StlSet<string>*

`-H` Use *SgiHashSet<string>*

(See the next section discussion for a discussion of these types.) If none of the above options are supplied, you are free to pick whichever representation you prefer as the default.

The output of the program's `-d` option will vary depending on which of the options above is chosen. If `-d` is specified with either the `-H` or `-T` options, the program should simply print

```
No statistics available.
```

in lieu of statistical output. If `-h` is specified, the statistics should be the same as in the previous assignment. For the `-t` option, you should print some sort of useful information about the structure of the tree. It is up to you to decide what information to print, how to calculate that information and to describe your choices in your `README` file.

If the program is given invalid arguments, it should produce appropriate error messages rather than crash. The choice of error message is up to you.

3 Required Classes and Functions

3.1 Required *AbstractSet* Abstract Template Class

Your code must provide and use the abstract template class *AbstractSet*. Note that you cannot create abstract classes, so you can only have pointers or references to, say, an *AbstractSet<string>*.

An instance of *AbstractSet*, on some type *T* (i.e., *AbstractSet<T>*), will specify the following (abstract) operations:

- A destructor that leaves the object in a sane (empty) state
- A member function *size_t* size() ***const*** that returns the size of the hash table
- A member function *void* insert(***const*** *T&*), where h.insert(x) adds x to the hash table, where the function's behavior is undefined if x has already been added to the table
- A member function *bool* exists(***const*** *T&*) ***const***, where h.exists(x) returns true if x is present in the hash table and false otherwise

(The type *T* is used for illustrative purposes—you may not use such a short name for the template type argument in your template class.)

All of the above functions must execute in $\mathrm{O}(\log n)$ expected amortized time, where n is the number of items in the set. In some cases, this bound may be loose.

You may add other (abstract) operations to this class if you wish (e.g., require a printStatistics member function).

You must write your code such that #include "abstractset.hpp" will declare and define this template class. We may test this class in isolation from the rest of your code.

3.2 Required *HashSet* Template Class

This class will be almost exactly the same as your *HashSet* class from the previous assignment, with one change: the class must now inherit from *AbstractSet* (i.e., for any *T*, *HashSet<T>* will be substituable for *AbstractSet<T>*).

You must write your code such that `#include "hashset.hpp"` will declare and define this template class. We may test this class in isolation from the rest of your code.

3.3 Required *TreeSet* Template Class

This class will represent a set using a binary search tree. Your *TreeSet* template class must inherit from *AbstractSet*, and must satisfy the complexity guarantees given in the specification for *AbstractSet*. The choice of binary search tree technique is up to you, provided you meet the complexity guarantees—we strongly recommend you choose a technique you understand and can code elegantly.[1] In particular, you cannot make any assumptions about the word order for words in the dictionary, only that each dictionary word occurs only once.

You may assume that all types used in instances of this class will support the `<` operation, much as your hash table code assumed the existence of a `hash` function.

You must write your code such that `#include "treeset.hpp"` will declare and define this template class. We may test this class in isolation from the rest of your code.

3.4 Required *StlSet* Template Class

This class will represent a set using the STL's *set* template class. In essence, it will be a fairly trivial wrapper around a *set* object, adapting the interface of *set* to the interface required by *AbstractSet*. As with the other classes, it must inherit from *AbstractSet*.

You must write your code such that `#include "stlset.hpp"` will declare and define this template class. We may test this class in isolation from the rest of your code.

3.5 Required *SgiHashSet* Template Class

This class will represent a set using SGI's *hash_set* extension to the STL (available using `#include <ext/hash_set>`). In essence, it will be a fairly trivial wrapper around a *hash_set* object, adapting the interface of *hash_set* to the interface required by *AbstractSet*. As with the other classes, it must inherit from *AbstractSet*.

You are not required to make the *hash_set* use your hash function, but you are welcome to find out how that is done (and thereby learn more about the joys of function objects, template specialization, and more).

You must write your code such that `#include "sgihashset.hpp"` will declare and define this template class. We may test this class in isolation from the rest of your code.

1. We will consider non-recursive code inelegant.

4 Testing and Submission

Once you have completed the programming portion of the assignment, you should compare the performance of your spelling checker using each of the set implementations and draw conclusions. It is sufficient to pick test cases and use the `time` command to compare execution times, but you may also use the profiling facilities present in `g++3` to get a more detailed picture of where the program spends its time.[2] You should include your conclusions in your `README` file, or (given that the `README` file is due two days before your code) your `NEWS` file.

All the usual rules for submission apply. You must, for example, provide a `Makefile` so that your `myspell` program will build from your submitted source when `make` is run.

5 Tricky Stuff

As usual, there are parts of this assignment that are tricky. Here are a few:

- You haven't written any code that uses inheritance, pure virtual functions, etc. So you should allow yourself some time to get familiar with coding in this way.
- You will need to restructure your spell-checking code from the previous assignment to make it use *AbstractSet<string>*.
- You cannot create an object belonging (only) to an abstract class. You can, however, have pointers or references to objects that can be viewed as belonging to an abstract class.

2. Profiling code is added by the compiler when you add the `-pg` option to the compiler flags. A summary of profiling information can then be printed after running the program by typing `gprof myspell | c++filt3`. See the `gprof` manual page for details.