CS1321 Homework 9*

Jeffrey D. Oldham

2000 Apr 19

Due Thursday, 2000 Apr 27, at the beginning of class.

Contents

1 Revisions

2000Apr25: In CS1321-2 today, it appeared the hash function scheme presented in lecture was broken. It is not. Read this exposition for an explanation why.

2 Reading

Read chapter 12.

3 Problem Statement

You and possibly a CS1321 classmate are to implement an open-chained hash table and incorporate it into a program to find duplicate WWW pages. You may solve the problem in any way except that you may not use any code available in written or electronic form.

prototype	example use	explanation
`insertPair`	`hashTable::insertPair p = insert( ... );`	`pair` of a boolean and a URL
`hashTable(void)`	`hashTable h;`	create a hash table with no entries
`insertPair insert(const URL & key);`	`h.insert( URL("www.google.com"));`	add the given URL to table if not already present
`bool query(const URL & key) const;`	`bool b = h.query(URL("www.google.com"));`	return true if and only if an == URL is in table
`bool remove(const URL & key);`	`h.remove( URL("www.google.com"))`	remove any == URL present in the table, returning `true` if and only if an == URL was found
`friend ostream& operator<<(ostream& out, const hashTable & h);`	`cout << h;`	print hash table's contents, one per line, in no specific order.

Table 1: `hashTable` Public Member Types, Functions, and Friends

4 C++ Programming Tip

Default function arguments can specify values for function arguments that usually have the same value. For example, the function `int foo(int x, int y = 3)` can be called using either `foo(4,5)` or `foo(4)`. In the former case, the parameter values are `x = 4` and `y = 5`. In the latter case, `y`'s parameter value is `3`. See also the textbook pp. 60–62.

5 Hash Tables

A *hash table* is a data structure supporting insertion, removal, and querying of elements in expected constant time by using a hash function. A *hash function* converts an element into a number specifying where the element should be stored. You are to implement an open-chained hash table storing `URL` objects, which conceptually are names of WWW pages. The algorithms and hash function to use were presented during lecture.

(The hash table described below has been designed to support finding URLs with the same contents. Other hash tables, e.g., the STL hash table, are similarly designed but details may vary. For example, most hash tables store key-value pairs.)

5.1 Operations

The `hashTable` class should support the operations listed in Table 1. These operations are similar to but not identical to these provided by the STL hash tables.

All `hashTable` operations use `==` or `!=` to compare keys.

The `insert` function ensures that the given key is in the hash table. If a matching key, i.e., one equivalent using `==`, is already in the hash table, the given key is not inserted. Thus, all keys in the hash table are always unique. `insert` returns a pair in which

- the first element indicates whether the key was previously present in the hash table. A `true` value indicates the key was not present prior to the insertion.
- the second element contains the matching key. This value is to be used only if a matching key was already present in the hash table before attempting the insertion.

5.2 A Hash Function

As discussed in lecture, our hash function converts a string, e.g., a URL's contents, to a position in the hash table's main vector. There exist many different hash functions, but we describe one that has proven to work well in practice.

Conceptually, our hash function

1. interprets the string as a base-256 number using the string's ASCII values. Actually, it may be easier to interpret it as a base-256 number with digits in reverse order. For example, the string "Bob" would be given

value $66 + 256 * (111 + 256 * 98)$. Computing this may be easiest using recursion on the given string. When converting a character to its ASCII value, be sure to convert it to an unsigned int, e.g., `static_cast<unsigned int>(character)`, to avoid negative numbers.

2. spreads out the strings' values using an irrational number. The number resulting from the conversion of the string is multiplied by an irrational number and then the digits to the left of the result's decimal point are dropped. Donald E. Knuth recommends using the golden ratio $(\sqrt{5} - 1)/2$ [Knu98, p. 517].

3. maps the resulting fraction to the range $\{0, 1, 2, \ldots, M - 1\}$, where M is the length of the hash table's main vector. We truncate the product of the fraction and M; i.e., we drop the nonintegral portion of the product.

In practice, converting the string into a base-256 number yields a very large number that can overflow the largest integer that most computers can store. Instead, we combine the first two steps. After converting each string's character into its ASCII code, multiply by the golden ratio. After each multiplication or addition drop the integral portion of the result. This will ensure that the numbers will always have reasonable size.

5.3 Resizing the Hash Table

A good hash function spreads out a hash table's contents, but, if any significant number of table positions have too many entries, using the hash table will take too much time. Thus, if the hash table becomes too full, we should enlarge the table. If twice the number of items in the hash table exceeds the size of the hash table's main vector, double the vector's size. If the number of items is less than one-eighth of the main vector's size, halve its size.

When resizing the hash table, every URL in the hash table must be rehashed since the hash function's values depend on the table's size. One implementation strategy s to insert the URLs into a newly created vector. Another strategy is to always have two vectors in the hash table object and a variable indicating which vector contains the URLs. To resize, the other vector is cleared, resized to the new size, the URLs are copied into it, and marked as the vector containing the URLs.

6 Finding Duplicate WWW Pages

Hash tables are commonly used as *dictionaries*, i.e., data structures supporting insertion, querying, and removal. They are also useful when determining duplication. For example, one problem faced by WWW search engines is identification of mirrored WWW pages, i.e., duplicates of other WWW pages, because it is not desirable to present the user with a long list of duplicate WWW pages. Directly comparing all pairs of WWW pages takes too long. Instead, WWW pages can be hashed and pages compared only if they hash to the same location. This reduces the running time from $O(n^2)$ to $O(n)$, where n is the number of WWW pages.

`findDuplicates.cc` uses a `hashTable` containing `URL`s to identify duplicates. Given the name of a file containing a list of URLs as its one command-line argument, the program identifies all duplicate WWW pages. For example, if three URLS "foo," "bar," and "baz" have the same WWW contents, two matches will be printed.

A `URL` (uniform resource locator) object stores a name. Conceptually, this is the name of a WWW page, ftp file, etc. To avoid the slowness of Trinity's Internet problems, we just use filenames, not WWW addresses.

Two URLs are `==` if and only if their files have exactly the same contents. Broken links, i.e., URLs without a file in the local directory, are not equal to any other URL. The hash table should hash the URL's contents, not its name.

As far as Jeffrey D. Oldham can tell, WWW search engines use technology similar to this homework to eliminate duplicates. The scheme, however, is fooled by "almost mirrors" which differ in small ways, e.g., hypertext links or use counters. Recent research shows that careful choice of hash functions can help automatically group related WWW pages. Other uses remain undiscovered.

7 Suggestions for Implementation

Using STL containers and functions may ease implementation. We suggest using a `vector` of `vector`s of `string`s. Let the *main vector* be the `vector` of `vector`s.

Useful `vector` functions include:

- `resize(size)`, which changes the number of items a vector can store,
- `erase(vector::iterator)` removes the item pointed to by the iterator, and

- `clear()`, which removes all the items in the vector.

Note that the `find` uses `==` to compare elements.

The number of distinct characters is `UCHAR_MAX`, which is defined in `limits.h`.

A possible order for implementation is

1. Implement the `hashTable` class using a hash function that always yields zero. Test the code for correctness.
2. Implement the more sophisticated hash function, making sure you print each hashed value so you can visually check for randomness. Use a hash table with a fixed reasonable size.
3. Implement hash table resizing. Try writing one function that is called whenever the number of items in the hash table changes. The function should then decide whether to resize the main vector or not.
4. Stress test your code.

The `cmp` UNIX command compares two specified filenames, printing whether the files differ or not. For more information, see the "diff" info pages accessible via "Control-h i" inside Emacs.

If you find using `URL`s confusing, try implementing `hashTable` storing another type, e.g., `key_type`, testing it. Then revise to use `URL`s.

8 What Files Do I Need?

We have provided code to find duplicate WWW pages and a `URL` class. Add code to the hash table to implement the hash table storing `URL`s. The Makefile may be useful.

9 Rules

9.1 Programming Rules

As for previous homeworks, working with other people during your planning phase is encouraged. For this homework, you are permitted to write the code, i.e., program, with one other person in CS1321. To learn the material, both of you should be actively involved in the programming. Failure to do so will almost certainly hurt your comprehension of the material in the rest of the course.

9.2 Submission Rules

Each one- or two-person team of programmers should submit only its completed implementation, consisting of the file `hashTable.h`. You do not need to send any other files. Please send only text documents, do *not* send Microsoft Word documents, PDF documents, HTML documents, etc. **Please include both your names and email addresses at the top of your program.**

We will test your program using our own input. Please be sure your code compiles without warning when using `g++ -Wall -pedantic`.

See the submission details for information how to email the programs. Note that the CS computer configuration was changed over spring break. If you want to receive an automated reply acknowledging your submission, please read the submission WWW page. If a team of two are in different sections, submit exactly once to one of the two permissible email addresses.

References

[Knu98] Donald E. Knuth. *Sorting and Searching*, volume 3 of *The Art of Computer Programming*. Addison-Wesley, Reading, MA, second edition, 1998.