INDUSTRIAL UNIVERSITY OF HOCHIMINH CITY

Data Structures and Algorithms

Lab 06: Hash Table

(Adapted from CS210, Department of Computer Science, University of Regina)

Highlights of This Lab:

- ☐ Definition of a Hash Table
- ☐ Application: Looking up Passwords

Lab Exercise: 

Click the little computer above for a detailed description.
For this excercise you will be asked to implement a password checking system to authenticate a users password.

---

1. Definition of a Hash Table

Before we get into the definition of Hash Tables, it is good to introduce WHY to use Hash tables.

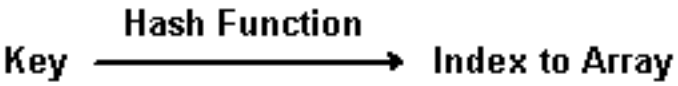
Hash tables are good for doing a quick search on things.

For instance if we have an array full of data (say 100 items). If we knew the position that a specific item is stored in an array, then we could quickly access it. For instance, we just happen to know that the item we want it is at position 3; I can apply: myitem=myarray[3];

With this, we don't have to search through each element in the array, we just access position 3.

The question is, how do we know that position 3 stores the data that we are interested in?

This is where hashing comes in handy. Given some key, we can apply a hash function to it to find an index or position that we want to access.



1.1 What is the hash function?

There are many different hash functions. Some hash functions will take an integer key and turn it into an index. A common one is the division method.

Let's learn through an example:

1.2 Division method (one hash method for integers)

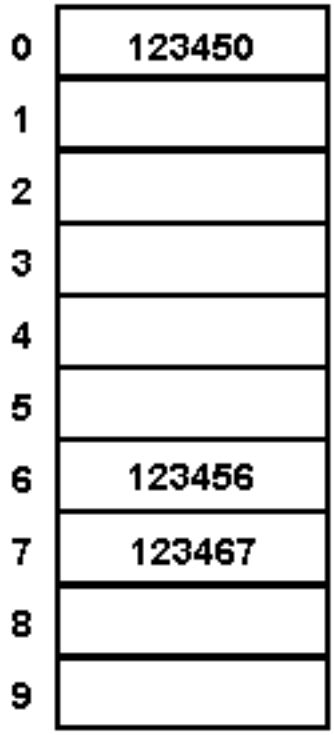
Let's say you had the following numbers or keys that you wanted to map into an array of 10 elements:

123456
123467
123450

To apply the division method, you could divide the number by 10 (or the maximum number of elements in the array) and use the remainder (the modulo) as an index. The following would result:

123456 % 10 = 6 (the remainder is 6 when dividing by 10)
123467 % 10 = 7 (the remainder is 7)
123450 % 10 = 0 (the remainder is 0)

These numbers would be inserted into the array at positions 6, 7, and 0 respectively. It might look something like this:



The important thing with the division method is that the **keys are integers.**

1.3 What happens when the keys aren't integers?

You have to apply another hash function to turn them into integers. Effectively, you get two hash functions in one:

1. function to get an integer
2. function to apply a hash method from above to get an index to an array

What do we mean that the keys aren't integers? Well, let's say that the keys are people's names. Such as:

Sarah Jones
Tony Balognie
Tom Katz

The goal is to type in one of these names and get an index to an array in order to access that information. How do we do this?
The hash function has to do two things:

1. Convert the names into integers

   For instance, we have a function which turns a string into an integer. The results will be as follows:
   Sarah Jones --> 1038
   Tony Balognie --> 1259
   Tom Katz --> 746

2. Apply a hash method to get an index

   We can now apply the division method to get an index for an array of 10 elements
   Sarah Jones --> 1038 % 10 --> 8
   Tony Balognie --> 1259 % 10 --> 9
   Tom Katz --> 746 % 10 --> 6

1.4 What would that look like in the array?

The array is known as a hash table. It stores the key (used to find the index) along with associated values. In the above example, we might have a hash table that looked something like this:

Again, the idea is that we will insert items into the hash table using the key and applying the hash function(s) to get the index.

A problem occurs when two keys yield the same index. For Instance, say we wanted to include:
John Smith --> 948 % 10 --> 8

We have a **collision** because Sarah Jones is already stored at array index 8.

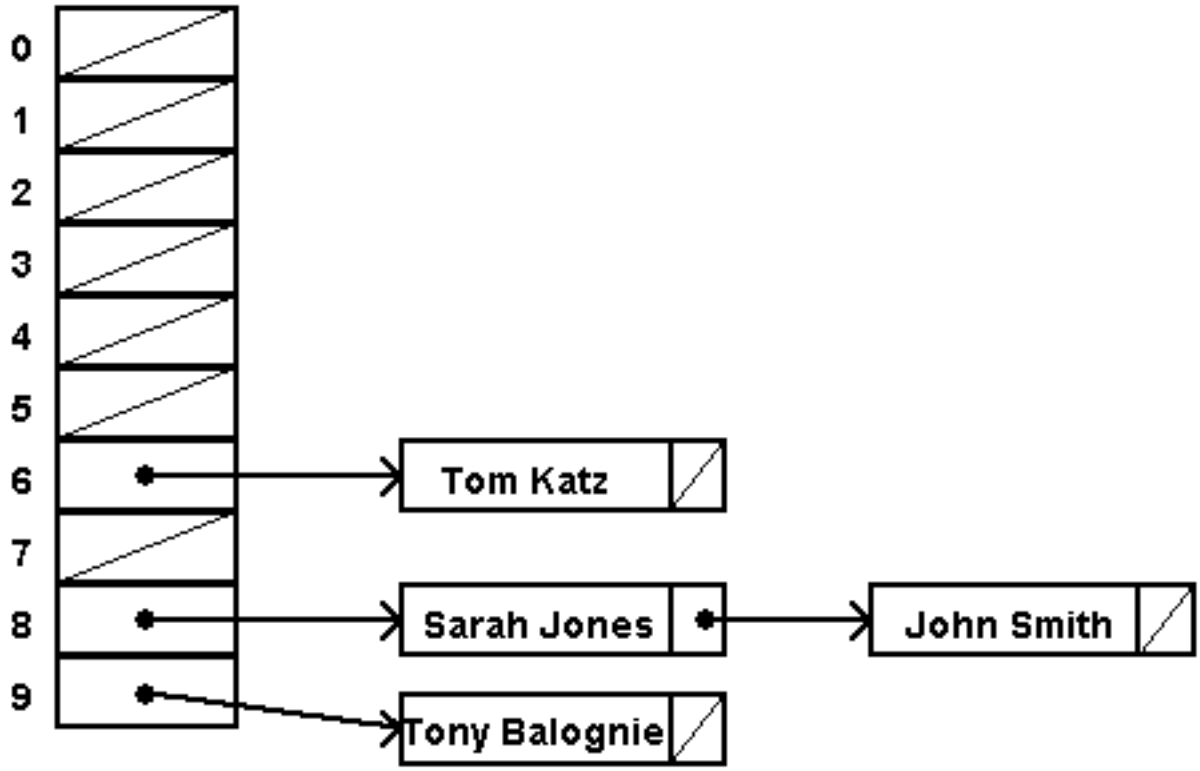
We need a method to resolve this. The resolution comes in how you create your hash table. There two major approaches given in the book:

1. Linear Probing
2. Chaining

The approach used in this lab is referred to as **chaining**.

The details are left as class material, but recognize that in **chaining** you have an array of linked lists. All the data in the "same link", have colliding index values.

Consider a diagram of the above example. Remember, there was a collision with Sarah Jones and John Smith. Notice that John Smith is "chained" or "linked" after Sarah Jones.



1.5 Applications of a Hash Table

Hash tables are good in situations where you have enormous amounts of data from which you would like to quickly search and retrieve information. A few typical hash table implementations would be in the following situations:

- For driver's license record's. With a hash table, you could quickly get information about the driver (ie. name, address, age) given the licence number.
- For compiler symbol tables. The compiler uses a symbol table to keep track of the user-defined symbols in a C++ program. This allows the compiler to quickly look up attributes associated with symbols (for example, variable names)
- For internet search engines. For more information, click **here**
- For telephone book databases. You could make use of a hash table implementatation to quickly look up John Smith's telephone number.
- For electronic library catalogs. Hash Table implementations allow for a fast find among the millions of materials stored in the library.
- For implementing passwords for systems with multiple users. Hash Tables allow for a fast retrieval of the password which corresponds to a given username.

1.6 Typical Operations of a Hash Table

The functions associated with our implementation of the hash table are the following:

- `bool isEmpty()`
  Returns `true` if the hash table is empty. Otherwise, returns `false`
- `bool isFull()`
  Returns `true` if the hash table is full. Otherwise, returns `false`
- `void insert (const DT &newDataItem)`
  Inserts `newDataItem` into the appropriate list in the hash table. The location (index) in the hash table is determined by the key and the hash function.
- `bool remove (KF searchkey)`
  Searches the hash table for the data item with the key `searchKey`. If the data item is found, then removes the data item and returns `true`. Otherwise, returns `false`.
- `bool retrieve (KF searchkey, DT &dataItem)`
  Searches the hash table for the data item with the key `searchKey`. If the data item is found, then copies the data item to `dataItem` and returns `true`. Otherwise, returns `false`.
- `void clear()`
  Removes all data items in the hash table.
- `void showStructure()`
  Outputs the data items in a hash table. If the hash table is empty, outputs, "Empty hash table". This is meant for testing/debugging purposes.

---

2. Application: Looking up Passwords

The following section outlines an algorithm for authenticating a user's password. Later, in the lab exercise, you will be given the skeleton code and asked to add lines to make it work.

One possible use for a hash table is to store computer user login usernames and passwords.

There are two major steps to this program:

1. The program will load username/password sets from the file `password.dat` and insert them into the hash table until the end of file is reached on `password.dat`. The password.dat file will look something like this with one username/password set per line:

```
2. jack    broken.crown
3. jill    tumblin'down
4. mary    contrary
5. bopeep  sheep!lost
```

6. The program will then present a login prompt, read one username, present a password prompt, and after looking up the username's password in the hash table, will print either "Authentication successful" or "Authentication failure". The output might look something like this:

```
7. Login: mary
8. Password: contrary
9. Authentication successful
10.
11.        Login: jim
12.        Password: contrary
13.        Authentication failure
14.
15.        Login: bopeep
16.        Password: sheeplost
17.        Authentication failure
```

Step 2 will be repeated until the end of the input data (EOF) is reached on the console input stream (cin). The EOF character on the PC's is the CTRL Z character.

Let's fill in some of the details:

To convert from a string to an integer, we can add the ascii value of each character together. For instance, mary's conversion from string to integer might look something like this:

```
109('m') + 97('a') + 114('r') + 121('y')=441
```

The code will look like this:

```
    int hash(const string str) const
    {
        int val = 0;

        for (int i=0; i<str.length(); i++)
            val += str[i];
        return val;
```

```
    }
```

We've converted the string to an integer, but we still need to convert the integer to an index. For an array of 10 elements we can divide by 10 and use the remainder as an index. Combining these two hash functions, we will get some code that looks like this:

```
   int index = dataItem.hash ( searchKey ) % 10;
```

Therefore mary's index will be:

```
 441 % 10 = 1.
```

---

3. Lab Exercise

- ☐ Get the files:

```
☐ Click here
☐
☐ To get the zipped files for this exercise
```

- ☐ Extract all of the files to the WORKAREA. Open the WORKAREA and double click on `exercise.sln`. This will open up the project. There are six files used in this program:
  - `hashtbl.cpp` and `hashtble.h` -- contain the implementation of the hashtable class
  - `listlnk.cpp` and `listlnk.h` -- contain the implementation of linked lists class
  - `login.cpp` -- the main program. This contains the `Password` structure, which is inserted into the hashtable.
  - `password.dat` -- contains all the users and corresponding passwords.

  Your primary tasks for this exercise are to edit the `login.cpp` to add in lines so that it does the following:

  5. insert passwords into the Hash Table
  6. retrieve one user's Password structure from the Hash Table
  7. check to see if the user is in the table and compare retrieved user password to input password, print "Authentication failure" or "Authentication successful"

  Steps include:

- ☐ Try to run this program. You should find that it will prompt you for "Login:" and "Password:" (type in random words at these prompts). You will notice that it continuely cycles around asking you for this information.
  To stop the program from running, at the "Login:" prompt, type CTRL and z (simultaneously) and then the Enter key .

- ☐ Add in a line to insert passwords into the table. Hint: notice that the name of the hashtable is `passwords` and that you want to `insert` a Password structure called `tempPass` into the hashtable.
- ☐ Add in a line to print out the hash table. Hint: the hashtable is `passwords` and there is a member function called `showStructure`.
- ☐ Build and Run this program. If all is working well, you should get some output that looks like this:

```
The Hash Table has the following entries
0: _

1: mary

2: _

3: _

4: _

5: bopeep

6: _

7: jill

8: _

9: jack

Login:
```

This shows the hash table that has resulted from inserting data from the `password.dat` file (mentioned in Section 2). Notice that mary is at index 1, just as we predicted (**in Section 2**).

- ☐ Add lines to compare the true password to the input password and print "Authentication failure" or "Authentication successful". Hint: Compare the input password (`pass`) to the password within the `tempPass` object (which has been retrieved).
- ☐ Build and Run your program. You should get results like the following:

```
Login: mary
Password: contrary
Authentication successful

Login: jim
Password: contrary
Authentication failure

Login: bopeep
Password: sheeplost
Authentication failure
```

- You now might want to play around with a couple of things and see what happens:
  - Modify the following line so that you have 8 elements in your hash table (instead of 10):

```
o  HashTbl<Password, string> passwords(10);
```

    What happens to the hashtable? Why?

  - Edit the `password.dat` file. This file has been added to the project (under "Resource Files" in the Solution Explorer). Double click on it to open.

    You can add usernames and passwords to test more. Try adding a username "ramy" (this has the same characters as mary, and, therefore, the same integer hash value)

Test Plan for "Login" Simulation Program

Login:	Password:	Authentication Predicted	Result
mary	contrary	successful	
jim	contrary	failure	
bopeep	sheeplost	failure	
bopeep	sheep!lost	successful	

---

4. Postlab Exercises

For postlab exercices, click here.
