STUDENT OUTLINE

Lesson 40 – Hash-Coded Data Storage

INTRODUCTION: In the labs in previous lessons you have searched a data file containing ID and inventory information in a variety of ways. Of the search algorithms studied, the fastest search algorithm was $O(\log_2 N)$ for a binary tree or binary search of an ordered array. It is possible to improve on these algorithms, reducing the order of searching to O(1), or linear function of file size. The data structure used to accomplish this is called a hash table, and the O(1) search is referred to as hashing.

An example of hashing could occur on the first day of school. When distributing student schedules, one long line is usually broken up into smaller lines, often by grade level. This concept could be extended to 26 lines. Students line up according to the first letter of their last name. This is what hash-coded data storage is about - breaking up and reorganizing one big list into many little lists.

The key topics for this lesson are:

A. Hashing Schemes
B. Dealing with Collisions
C. Order of a Hash-Coded Data Search
D. `HashSet` and `HashMap`

VOCABULARY:

HASHING	HASH KEY
HASH TABLE	COLLISIONS

DISCUSSION: A. Hashing Schemes

1. The example of distributing student schedules illustrates a natural means of hashing. The process can be simplified by organizing the schedules into piles by the first letter of the last name.

2. A hashing scheme involves converting a key piece of information into a specific location, thus reducing the amount of searching in the data structure. Instead of working with the entire list or tree of data, the hashing scheme tells you exactly where to store that data or search for it.

UPC Version A

12345 67890

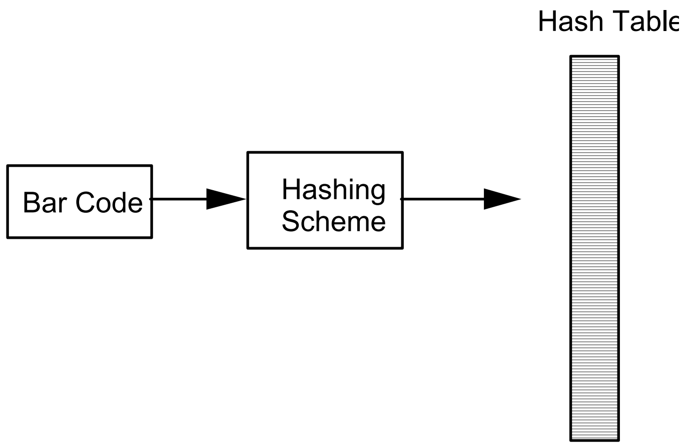
3. One important bar code system used by retail stores is the UPC A code*, which involves a sequence of 10 digits. This system provides for 10 billion different possible products, 0-9,999,999,999 (which equals $10^{10} - 1$). For quick access, an array of 10 billion locations would be nice, but wasteful in terms of computer memory. Since it is unlikely that a store would carry such a huge number of items, we need a system to store a list of products in a reasonably sized array.

   *Bar code graphic and explanation of how to read bar codes at:
   http://www.adams1.com/pub/russadam/upccode.html,

4. A cash register using a bar code scanner needs a very quick response time when an item is scanned. The 10-digit bar code is read, the item is searched for in the store's database, and the price is returned to that register. While searching algorithms of the order ($\log_2 N$) are relatively fast, we seek an even faster algorithm.

5. Suppose a store maintains a database of 10,000 bar codes out of the possible 10 billion different values. The values will be stored in an array called a hash table. Because an array has direct random access to every cell, using a hashing scheme will give very fast access to the desired item. The hash table is usually sized about 1.5 to 2.0 times as big as the maximum number of values stored. (The reason for this sizing will be apparent in a few sections.) Therefore we will create an array with 15,000 locations.

6. The hashing scheme will tell us where item XXXXX XXXXX is stored out of the 15,000 locations. A hashing algorithm is a sequence of steps, usually mathematical in nature, which converts a key field of information into a location in the hash table.



7. These "key-to-address transformations" are called hashing functions or hashing algorithms. When the key is composed of character data, a numerical equivalent such as the ASCII code is used so that mathematical processing can take place. Bar codes are numbers, so conversion is not necessary. Some common hash functions are:

   a. Division. The key is subject to modulo division by an integer (often a prime) equal to or slightly smaller than the desired size of the array. The result of the division determines which short list to work with in the hash table.

   b. Midsquare. The key is squared and the digits in the middle are retained for the address. This probably would not work well with bar codes because they are such large numbers.

   c. Folding. The key is divided into several parts, each of which are combined and processed to give an address. For example:

      If the bar code = 70662 11001

      1) group into pairs: 70 66 21 10 01

      2) multiply the first three numbers together:

         70 x 66 x 21 = 97020

      3) add this number to the last two numbers:

         97020 + 10 + 01 = 97031

      4) find the remainder of modulo division by 14983 (the largest prime less than 15000):

         97031 % 14983 = 7133

      5) address 7133 is the location to store bar code 70662 11001

8. In the address 7133 will be stored all the fields related to this item, such as price and name of the item.

9. It is important to develop a good hashing function that avoids collisions in the hash table. Even using a prime number for the divisor, it is possible for two bar codes to result in the same address - 7133. To reduce the chances of such a "collision", the hash table is sized about 1.5-2.0 times the number of expected data. If the hash table in our example were sized at 10,000, the number of items in the database, the likelihood of collisions is increased. We are trying to balance the need for decreasing the number of collisions against memory limitations, hence the recommended sizing.

10. This advance sizing of the hash table affects the mathematics of the hashing algorithm; therefore the programmer must have a very clear idea of the number of data to be stored. The number of items must be known in advance and this number must be fairly constant during the life of the program. This limits the use of hashing to certain situations. If the number of data is unknown or varies greatly, hashing is inappropriate.

B. Dealing with Collisions

1. There are two methods of creating multiple storage locations for the same address in the hash table. One solution involves a matrix, while the other uses dynamic linked lists.

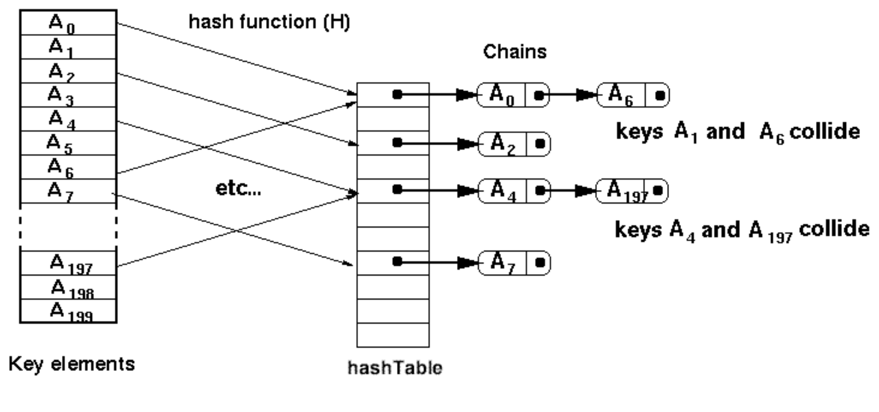
2. To implement the hash table as a matrix, an estimate of the maximum number of collisions at any one address must be made. The second dimension is sized accordingly. Suppose that number is estimated as 5:

```
Item[][] table = new Item[15000][5];
```

3. This method has some major drawbacks. The size of this data structure has suddenly increased by a factor of five. The above table will have 75,000 cells, many of which will be empty. Also, what happens if a location must deal with more than 5 collisions?

4. A dynamic solution, referred to as chaining, is much better. The linked lists will only increase as values are stored in that location in the hash table. In this version, the hash table will be an array of object references.

```
ListNode[] hashTable = new ListNode[MAX];
```



5. The order of the values in the linked lists in unimportant. If the hashing scheme is a good one, the number of collisions will be minimal.

C. Order of a Hash-Coded Data Search

1. After scanning an item at a cash register, the number of steps required to find the price is constant:

    a. Hash the bar code value and get the hash table location.

    b. Go to that location in the hash table, traverse the linked list until you find the item.

    c. Return the price.

2. The number of steps in this algorithm is constant. The hashing scheme tells the program exactly where to look in the hash table; therefore this type of search is independent of the amount of data stored. We categorize this type of algorithm as constant time, or O(1).

3. If the linked lists get lengthy, this could add a few undesirable extra steps to the process. A good hashing scheme will minimize the length of the longest list.

4. An interesting alternative to linked lists is the use of ordered binary trees to deal with collisions. For example, the hash table could consist of 10,000 potential binary trees, each ordered by a key field.

5. Remember that determining the order of an algorithm is only a categorization, not an exact calculation of the number of steps. A hashing scheme will always take more than one step, but the number of steps is independent of the size of the data set, hence we call it O(1).

D. `HashSet` and `HashMap`

1. The `HashSet` and `HashMap` classes are implementations of the `Set` and `Map` interfaces from the Java standard class Library. A hash table is used to store their elements.

2. The methods from the `HashSet` and `HashMap` that are included in the AP Subset are shown below.

   Methods of the `HashSet` Class included in the AP Subset[*]

```
// Adds the specified element to this set if it is not
// already present. Returns true if the set did not already
// contain the specified element.
boolean add(Object obj);

// Returns true if this set contains the specified element.
boolean contains(Object obj);

// Returns an iterator over the elements in this set.
// The elements are returned in no particular order.
Iterator iterator()

// Removes the specified element from this set if it is
// present. Returns true if the set contained the specified
// element.
boolean remove(Object obj);

// Returns the number of elements in this set.
int size();
```

   Methods of the `HashMap` Class included in the AP Subset[*]

```
// Returns true if this map contains a mapping for the
// specified key.
boolean containsKey(Object key);

// Returns the value to which the specified key is mapped,
// or null if the map contains no mapping for this key.
boolean get(Object key);

// Returns a set containing the keys in this map.
Set keySet()

// Adds the key-value pair to this map. Returns the previous
// value associated with specified key, or null if there was
// no mapping for key.
boolean put(Object key, Object value);

// Returns the number of key-value pairs in this map.
int size();
```

3. In the `HashSet` class, a hash of the element is used to find its location in the hashtable. In the `HashMap` class, a hash of the key is used. The `hashCode` method, which exists on all objects, calculates the hash code.

4. The basic operations on `HashSet` and `HashMap` objects run in constant, `O(1)`, time due to the hash table implementation employed by the class.

5. The iterator that is returned by the `iterator` method of `HashSet` does not order the objects returned.

[*] Adapted from the College Board's *AP Computer Science AB: Implementation Classes and Interfaces*.

SUMMARY/ REVIEW: Hashing is a great strategy for storing and searching information, especially where speed is a priority. In the hashing approach, the key is converted by some hashing function into an integer that is used as an index into a hash table. Different keys may be hashed into the same index, causing collisions. The performance and space requirements for hash table vary depending on the implementation and collision resolution method. In the best case a hash table provides O(1) access to data, but the performance deteriorates with a lot of collisions. In the lab exercise, you will implement a hash coded data storage scheme and determine its efficiency.

ASSIGNMENT: Lab Exercise L.A.40.1, *Hashing*

LAB EXERCISE

Hashing

Background:

A larger version of the store inventory data file (*file400.txt*) will be used in this lab exercise. As you may recall, the text file consisted of two fields, *id* and *inv*. The *id* values on disk will range from 1-20000 and the *inv* fields from 1-100. There are no duplicate `id` values in the data file. Every `id` value is unique in the file and the number of store items in this problem is 400. Use this value to size your hash table.

Assignment:

1. Write a program that uses a hash-coded data storage method to store the 400 items in (*file400.txt*). You can start with an earlier program such as the binary search lab to speed up your work.

2. The method of dealing with collisions should be the dynamic linked list version.

3. You should develop your own hashing scheme to take the key field (`id`) and determine the correct address.

4. Test your new data structure and algorithm with some sample searches. The program should prompt you for an `id` value and return the `inv` amount or a message that the `id` does not exist. Your instructor will specify some sample `id` values to test out.

5. Your program must analyze the efficiency of your hashing scheme by determining the following statistics about your hash table:

   a. The % of **`null`** pointers in the hash table.
   b. The average length of linked lists.
   c. The longest linked list in the hash table.

After seeing these results, you might want to try to improve upon your hashing scheme if the number of collisions is excessive.

Instructions:

1. Turn in your source code and a run output of the following results:

   a. The searches

   b. The statistical analysis of the hash table.