Advanced Data Structures
Week 1 Notes for CMP SCI 282

Week 1 Notes

New Feature - Integrated Course Notes and AJAX Exercises

This is a new feature developed for my 2009 Flex project. Integrated in these notes is a set of online exercises. As you read the notes, fill in the exercises to confirm your understanding of the topics. At the end of the notes, submit the exercises for grading. Please start with your name and email address:

First Name Nick Last Name Rebhun Email Address nrfactor@gmail.com

Data Structure - n : the organization of data and its storage allocations in a computer

How the data is organized in computer memory

- RAM – Random access memory, main memory, transient
- disk storage – Hard drive, secondary storage, permanent

This class is the follow-on to Computer Science 182
Data Structures and Program Design

182 concentrates on main memory (RAM) data structures

282 concentrates on disk storage (hard drive) based data structures

Very different approach between RAM and external storage

- RAM is very fast, but space is limited
- Disk storage very slow, but space is abundant

Obviously, RAM vs. DISK has a huge effect on 'which' data structures to use

Some data structures will be the same as 182, others that we study in 282 will be different

What is a database?

According to dictionary.com:

database - Computer Science
n. also data base

A collection of data arranged for ease and speed of search and retrieval.

For all **practical purposes** a database is a data structure stored to disk!

Think about it, take most any data structure from 182, sorted array, binary tree, hash table and use it to store data on disk and you get a database.

A lot more goes into a database, but at its core, it is a data structure stored on disk.

A big part of this class is choosing the right data structure for your project!

The topic, what data structure to use when presented with a problem to solve, is far more important then students realize.

Choose the right data structure, the problem is easy to solve, the code is simple
Choose the wrong one, the data does not 'match' the structure, the code is hard

The structure you choose, will GREATLY affect your program design

Driving directions - Start in the wrong direction, drive an hour, you've wasted 2 hours. Why?
Software development is similar, start with the wrong data structure, try to 'save it'

Business - Start your own company - Hire the wrong people - Who do they hire?

EBay vs. EToys

(if time, compare the two)

Chapter 11 - Hash Tables

A Hash Table is a data structure designed for fast insertion and retrieval of data

Almost unbelievably fast, Big O notation: O(1) // that is fast folks!

How do we find the data we are looking for?

Follow the yellow brick road, Follow the yellow brick road..... --- The movie is easy, but who said it?

First and Foremost
To access data within a hash table, calculate the location of the data then store or retrieve it.

With a hash table, use a little math (which computers do very fast), the result of the math is the location of the data in the hash table.

1. A hash table is based on an array data structure.
2. Data is stored in the hash table array using a key.
3. To place data into a hash table, use the key to calculate an index in the array and store the data there
4. To retrieve that data, you need the SAME key, do the SAME calculation, to find the SAME array index

Just like the binary tree uses a key to store/find data in the tree using the less then to the right, greater then to the left rule

A hash table needs a key to calculate the location in the array for placing the data, then uses the key again to find the SAME location when it is time to retrieve the data

Without the key, do not bother looking for something in a hash table

How do you calculate a location?

Lets look at the number 123, to put it another way 123 = 100 + 20 + 3

or 123 = $1 * 100 + 2 * 10 + 3 * 1$

or 123 = $1 * 10^2 + 2 * 10^1 + 3 * 10^0$

Since there are 10 decimal digits (0-9), each 'place' in the number is the number times a power of 10
No matter what combination of numbers you substitute for each digit in 123 you will always come up with a unique number (000 - 999)

To be clear, the above formula multiplies by a power of 10, **because there are 10 decimal digits!** If we want to convert text in a similar manner, we would multiply by powers of 26, **because there are 26 letters in the alphabet!**

Fill in the answers to the following Hash calculation questions:

26^0 is 1, 26^1 is 26, 26^2 is 676, 26^3 is 17576 **, 26^4 is** 456976 **, 26^5 is** 11881376

Here is how we convert letters for hashing: Since there are 26 letters in the alphabet we take each letter and multiply it by a 'power' of 26. Assign a=0, b=1, c=2, ... ,z=25 and abc can be converted to a number like so:

	abc = $a * 26^2 + b * 26^1 + c * 26^0$
or	abc = $0 * 26^2 + 1 * 26^1 + 2 * 26^0$
or	abc = $0 * 676 + 1 * 26 + 2 * 1$
or	abc = $0 + 26 + 2$
or	abc = 28

No matter what combination of lower case letters you substitute for each

digit in abc you will always come up with a unique number (0000 - 17575)

Note: The book allows a space ' ' character in the key, so it does the calculation based on powers of 27

This calculation depends on the number of different characters in the key.

If the key can contain up to 62 characters, you have to use powers of 62
26 upper case + 26 lower case + 10 numeric digits = 62

A text string can be converted to a unique number with a simple formula, assuming a = 0, b = 1, c = 2, ... z = 25, the string 'cat' can be converted like this:

$$'c' * 26^2 + 'a' * 26^1 + 't' * 26^0$$

$$2 * 26^2 + 0 * 26^1 + 19 * 26^0$$

$$2 * 676 + 0 * 26 + 19 * 1$$

$$1352 + 0 + 19 = 1371$$

Convert the string 'pear' to a number using the above formula: 266361
Convert the string 'lime' to a number using the above formula: 199060
Convert the string 'plum' to a number using the above formula: 271608

Where does the term '**hash**' come from?

Lets say you only want to store 'legal' 3 letter words. That is, cat, fat, rat and sat, but not aaz, zzb etc.

You need a LOT fewer array elements then an array of size 17575, perhaps there are less then 1000 3 letter words.

You would only need 1000 slots in the array to hold them all. How do you '**map**' a large number like 17575 into a smaller space like 1000 array slots.

The answer is the modulus function (which uses the % operator)

When ever you do integer division arithmetic you ALWAYS get 2 answers, how many times the numbers divide into each other, and what is left over, the remainder

7 / 2 = 3 // With a remainder of 1

15 / 6 = 2 // With a remainder of 3

7 / 8 = 0 // With a remainder of 7

The modulus operator % calculates the remainder directly, ignoring how many times the numbers divide

7 % 2 = 1

15 % 6 = 3

7 % 8 = 7

The modulus (or %) function has a very interesting property. The result is **NEVER, EVER, NOT POSSIBLY, EVER** bigger then the second operand

Think about it, someNumber % 3 = someAnswer

someAnswer could never be larger then 3!

3 % 3 = 0
4 % 3 = 1
5 % 3 = 2
6 % 3 = 0
7 % 3 = 1
8 % 3 = 2

9 % 3 = 0 // as soon as the first number gets larger, the remainder resets to 0

Think about it,
someNumber % 3 = 3 // can't happen, the number of times it divides will go up by one, the real remainder is 0
someNumber % 3 = 4 // can't happen, the number of times it divides will go up by one, the real remainder is 1

So if you were to execute someNumber % 1000 = someAnswer

someAnswer is 100% (pun intended) guarenteed to be between the range 0 and 999
which makes it a perfect value to use as the index in an array of 1000 items

So where does the 'hash' in hash table come from?

Hashing is the term for converting a key to a number **AND** then mapping that number to a suitable range for use as a subscript in an array.

For example, 'cat' converts to the value 1371 **AND** 1371 % 1000 = 371

When you want to store 'cat' in the hash table, place it in slot 371 of the array.

When you want to retrieve 'cat' from the hash table, the **SAME** calculation takes you right back to slot 371

A ***hash function*** takes a key as input and returns the subscript in the array where the data (and key) are stored.

Nice round numbers like 1000 are great for showing calculations (easy math) but the do not produce the best results in a hash function. Prime numbers (can only be divided by 1 and itself) produce much better results. Prime values like 1009 or 1013 should be used instead of 1000 in a 'real' hash function.

The 'hash' in Hash Table refers to mapping a key/number to a particular slot in an array. This requires using the modulus operator '%' and a prime number to calculate the proper slot in the array:

```
26   %   11  =  4    Why?  26 / 11 = 2, remainder 4   or  26 - ( 11 * 2 ) = 4
95   %   17  =  10   Why?  95 / 17 = 5, remainder 10  or  95 - ( 17 * 5 ) = 10
```

Hash (or map) the large number 123905 to an array slot using the prime number 211 : 48

Hash (or map) the large number 457591 to an array slot using the prime number 229 : 49

A hash table is a blazing fast data structure. The result of the calculation, takes you straight to where your data is stored!

Lets say I store the value 'cat' some where in a **normal** unsorted array with 1000000 slots. To find the key 'cat', I must search 300000 or 500000 or 700000 slots of the array? (average search is 500000 slots N/2)

Now store the value 'cat' some where in a **hash table** array with 1000000 slots. To find the key 'cat', I convert it to a number and do a (number % 1000000) to go straight to the array slot. In most cases, one array slot is examined (not 500000).

What if I hid a CD-ROM disk somewhere in this classroom, how long would it take you to find it? 3 maybe 4 hours?
What if I hid a CD-ROM disk under a printer in this classroom, how long would it take you to find it? 3 maybe 4 minutes?
That is what the hash function does, it takes you straight to the location of the data in the hash table.

Now there is an official notation for comparing algorithms

Big O Notation - as in Order of

It is a ***relative*** measure of an algorithm/data structures performance as N increases

Hash Tables are Big O notation: **O(1)** // that is fast!

When N = 1000, convert key to a number % 1000 = location

When N = 10000, convert key to a number % 10000 = location

When N = 100000, convert key to a number % 100000 = location

As N increases, the time needed to do the calculation and locate the data, is exactly the same, that is what **O(1)** means

Unordered Array searches are Big O notation: **O(N)** // not so fast!

When N = 1000, average 500 slots = location

When N = 10000, average 5000 slots = location

When N = 100000, average 50000 slots = location

As N increases doubles in size, the search average doubles, that is what **O(N)** means

The bigger the unordered array, the slower the search. Hash tables will NOT slow down!

Run the Applet now

So if hash tables are so fast, why use anything else?

Disadvantages of Hash Tables

1. You can not access the data in a sorted order

2. You must allocated extra storage for the array

Disadvantage 1
Recall the sorted array data structure, you could scan the array and see the data in a a sorted order.
If the key was a persons last name, the data in the array, names from Anderson,... to ...,Ziegfried would be in exact order

There is NO such access to a hash table, if you programmed it yourself, you could access each slot in the array, but the data will come out in a pseudo random order. Some implementations allow NO access to the array.

If you want the data sorted, you then have to do a very time consuming sort.

Some have created hybrid data structures that combine a hash table and a sorted list or array. Works if you don't insert a whole lot, but quickly slows down if you do.

Bank example - Tellers want fast look up of accounts (hash table), but at the end of the month, how do you process each statement in order (by name, address, etc.)

Disadvantage 2
The extra space needed is the second major disadvantage. There is no such thing as a perfect hash function.

If you need to store 1000 items in a hash table, you need to allocate the array at least twice that size.

Hash functions will, and often do, hash to the same location in the array.

For example the key, "cat" and key "hat" both want to go in slot 371 of the array. This is called a collision

Is #2 a 'real' disadvantage? For a disk based data structure, hard drives are very large and very cheap

Space is abundant and does not cost very much

When it comes to disk based hash tables, there is only 1 (or maybe 1 1/2) disadvantages

---

Using the above formulas, convert a three letter word to a large number, and then hash (or map) that number to an array slot using the modulus of a prime number:

The three letter word is 'tom' convert it to a number and map it to an array slot using the prime number 227 :
54

The three letter word is 'get' convert it to a number and map it to an array slot using the prime number 223 :
165

Submit Integrated AJAX Exercises

Results will appear below

Name: Nick Rebhun, nrfactor@gmail.com

Here are the results:

Your answer to 26^3: 17576 is correct

Your answer to 26^4: 456976 is correct

Your answer to 26^5: 11881376 is correct

Your answer, pear converts to: 266361 is correct

Your answer, lime converts to: 199060 is correct

Your answer, plum converts to: 271608 is correct

Your answer, 123905 modulus (%) 211: 48 is correct

Your answer, 457591 modulus (%) 229: 49 is correct

Your answer, hash/map tom using prime 227: 54 is correct

Your answer, hash/map get using prime 223: 165 is correct

Your point total is 10 out of 10

END Week 1 Notes

End of Week 1 Notes