



2007-04-10
Lecture #16
(Material on Test 3)

Today's Outline
Hash Tables
Random # Generators

Hash Tables

- some hash function determines the location of the data

Direct hashing

- data structure must have an element for every key
 - example: an array

One Index for every element in the array

Collisions

- **This is what it's called when we have 2 elements for 1 spot**

How do we get a key?

- **We could just number every element**
- **Students have SSN or Titan #'s**
- **Dictionary Words – sum of all ASCII values in word**
- **Dictionary Words – ASCII value of just the first letter**
- **many more ...**

What's an example of a hash function?

- module (remainder) is the most common

KEY % LIST_SIZE = ADDRESS

Note: Mathematically it has been shown that to achieve the least amount of collisions (collisions are bad) your list size should be a prime number

The first 30 prime numbers are 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97, 101, 103, 107, 109, and 113

Example:

List Size = 5

Hash Function = ASCII(KEY) % 5

Key = ASCII Value of Character

Practice hashing these data elements:

A, j, k, B, c, a, b, d

Other Hash Function ideas (note: there are unlimited ideas)

Digit Extraction

- from SSN or Titan #, always pick the 5th, 6th, 8th, and 10th digits

Key = 999129111212

Hash(Key) = 2912

Other Hash Function ideas

Mid Square

- Square the # ... pick the middle digits

Example:

Take first 3 digits and square

$$379452: 379 * 379 = 143641 \Rightarrow 364$$

$$121267: 121 * 121 = 014641 \Rightarrow 464$$

Random Number Generators

Example C++ code:

```
/* srand example */  
#include <stdio.h>  
#include <stdlib.h>  
#include <time.h>  
  
int main ()  
{  
    printf ("First number: %d\n", rand() % 100);  
    srand ( time(NULL) );  
    printf ("Random number: %d\n", rand() % 100);  
    srand ( 1 );  
    printf ("Again the first number: %d\n", rand() %100);  
    return 0;  
}
```

Multiple things to notice ...

Rand % 100

What's up with that?

srand(SEED)

How does that work?

Ah-Ha, it's actually only Pseudo Random!

Another Hash Function Idea

Pseudo Random Method

- use the key as the seed to get a pseudo random #, then % it

srand(KEY)

int address = rand() % KEY

Collision Resolution

- recall, collisions are when 2 items want to go in the same spot

Rule of Thumb ... hashed list shouldn't be more than 75% full, otherwise there will be way too many collisions ... and performance degrades

2 main ideas

- Use main array to store the collisions
- Use separate arrays / data structures to store the collisions

Note: there can also be many varied combinations

Using the Main Area to resolve collisions

called ... Open Addressing

Three basic ideas

1.) Linear Probe

- if you hit a collision, climb down until you find an open spot

2.) Quadratic Probe

- if you hit a collision, add 1 squared, 2 squared, 3 squared, etc.
until you find an open spot

3.) Double Hashing

- have a 2nd, backup hash function ... thus you re-hash it and use that address (note: if this still collides ... then you must linear probe or quadratic probe anyways)

Separate areas for collisions

Linked Lists

- **Easy implementation ... each element in the array holds a pointer to a linked list node ... empty list if empty**
- **Works best with LIFO (insert in front ... so it's fast)**

Buckets

- **Each address actually holds multiple data items**
 - **this can be bad, because of space it takes up**
 - **and it doesn't resolve collisions ... just delays them**
 - **still will have to use a linked list or something**

Other ideas ...

- **You could use a Binary Search Tree instead of a Linked Lists**
- **Just remember, if your collisions resolution is getting too complicated, then perhaps your hash function isn't too good**