ECE 2574 Introduction to Data Structures and Algorithms

36: Hash Tables

Chris Wyatt
Electrical and Computer Engineering
Virginia Tech

Dictionaries

A balanced tree can be used to very efficiently store and retrieve information.

Example: in a 10,000 word dictionary based on a Red-Black tree takes 13-14 comparisons on average to insert/find/retrieve.

In-order traversals are still linear.

Dictionaries

What if we need to retrieve faster?

Example: File System

Consider a simple disk, modeled as an array. We can move to a specific index to start reading the file contents.

How could we find the index where the file "myfile.txt" is found?

Simple File System

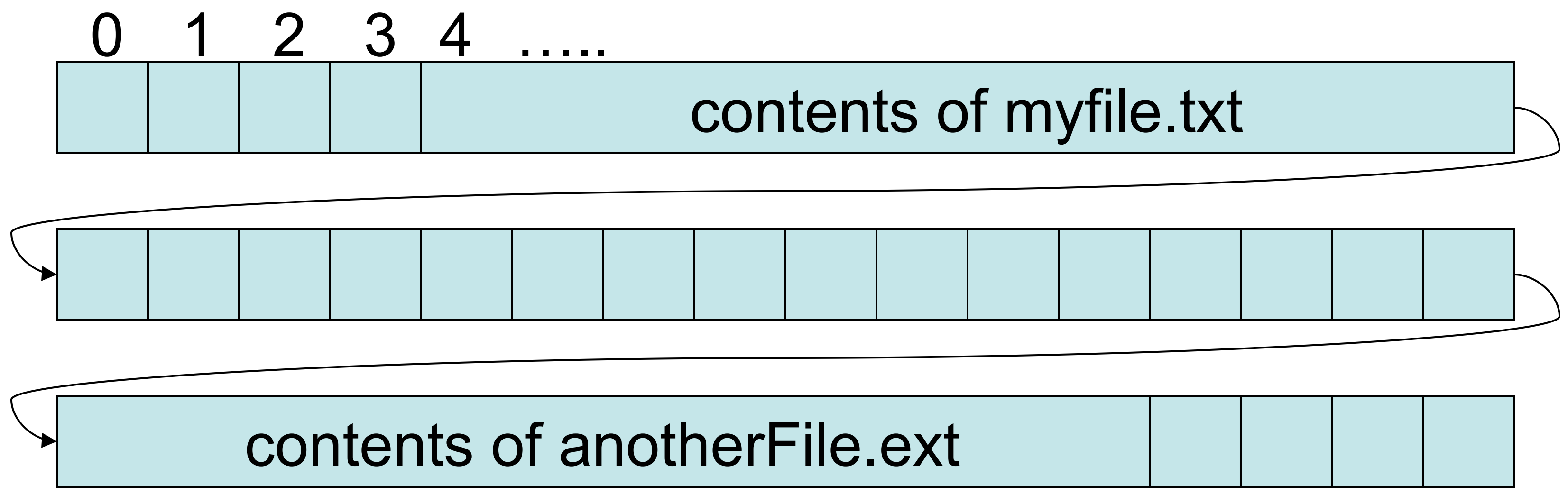
Filenames (no directories):

myfile.txt

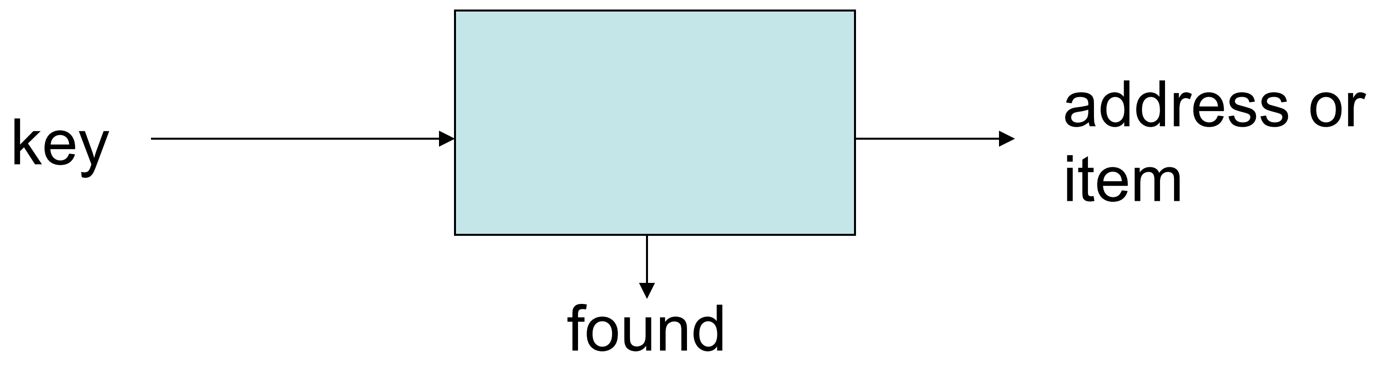
anotherFile.ext

Given file name we want to locate where is starts fast.

Disk



Block diagram of the retrieve task



Think about the box as an address calculator, it takes a key and maps it to an address where the item is stored.

$$\text{address} = \underset{\substack{\uparrow \\ \text{hash function}}}{h}(\text{key})$$

Example Uses of hashes

General dictionaries

Cryptography and Passwords (example: SHA)

Error correction (example: CRC)

Identification and verification (example: MD5)

Media identification / retrieval (name that tune)

Finding objects (geometric hashing)

Retrieve using a hash function

```
retrieve(in key:keyType,  out item:itemType): bool

    itemType loc = hash(key)
    if(loc.key != key)
        return false
    else
        item = loc.item
        return true
    endif
```

Insert is as easy

```
insert(in key:keyType,  out item:itemType)

    itemType loc = hash(key)

    loc.item = item
```

Two fundamental questions

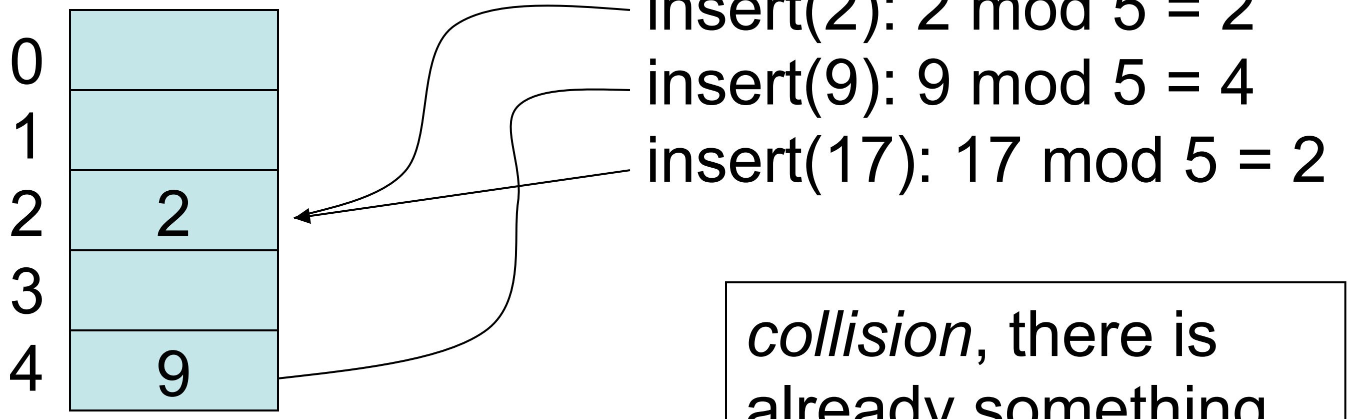
1. How to determine the hash function
   - lots of options
   - a bit of a black art (requires experimentation)

2. How to store the items in memory
   - using an array with a hash function is called a *hash table*

How to determine the hash function

Simple example: Given an array[0:m-1] and key, k, a positive integer

h(k) = k mod m



collision, there is already something in slot 2

Perfect hash function

A hash that has no collisions is called *perfect*.

There are actually tools to help design perfect hash functions, *if you know all the strings in advance*.

Example: gperf

http://www.gnu.org/software/gperf/

Collisions

What if you don't know the possible items ahead of time? - no perfect hash may exist.

There are two basic approaches to resolving collisions:

1. open addressing
2. chaining

Open Addressing

In open addressing, we move on to another slot. If that one is full, we move to another, ….

This is called *probing*. We probe for an empty slot.

(note this probe sequence must be repeatable)

Linear probing is the simplest:

```
index = h(key)
while array[index] is not full
    index = index + 1 mod array.size
endwhile
```

How do you know if an index if full?

Some possibilities:

Reserve an item value that indicates empty.

Each array entry is a struct with item and empty fields

Array is an array of pointers, with NULL indicating empty.

In class exercise

For a hash table of size 11 and a hash function

h(k) = k mod 11

use linear probing to insert keys 2,8,12,19,20,32,11

Quadratic Probing

To reduce clustering in the hash table, you can use quadratic probing

```
index = h(key)
probe = 1
while array[index] is not full
    index = h(key) + probe*probe mod array.size
    probe += 1
endwhile
```

Another approach: rehashing

If there is a collision, hash again using a different function to obtain the linear probe step size

Example: for a table of size 11

h1(k) = k mod 11, this is the primary hash

h2(k) = 7 - (k mod 7), this is the secondary hash

Note: h2(k) can't be zero and h2(k) can't equal h1(k)

In class exercise

For a hash table of size 11 and hash functions

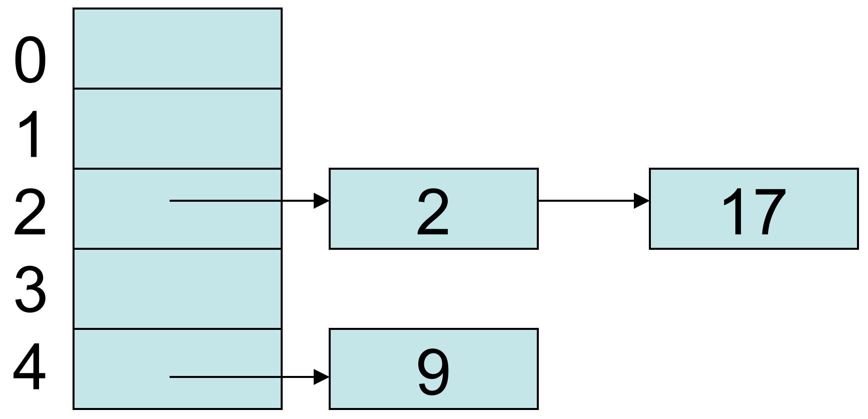
$h2(k) = 7 - (k \bmod 7)$

use rehashing to insert keys 2,8,12,19,20,32,11

2nd approach to collisions: chaining

Make the hash table an array of linked lists.

insert(2): 2 mod 5 = 2
insert(9): 9 mod 5 = 4
insert(17): 17 mod 5 = 2



In class exercise

For a hash table of size 11 and hash function

h1(k) = k mod 11

use chaining to insert keys 2,8,12,19,20,32,11

(sketch the linked lists)

Choosing (and designing) hash functions

sizes.

A hash function should be

- fast to compute
- distribute data evenly through the table (to prevent collisions)

reaches about 2/3 of m, hashing becomes inefficient.

Some well known hash functions

Robert Sedgwicks (RS) hash

```
unsigned int RSHash(const std::string& str)
```

Robert Sedgwicks (RS) hash

```
unsigned int RSHash(const std::string& str)
```

```
{
   unsigned int b    = 378551;
   unsigned int a    = 63689;
   unsigned int hash = 0;

   for(std::size_t i = 0; i < str.length(); i++)
   {
      hash = hash * a + str[i];
      a    = a * b;
   }
```

Some well known hash functions

```
unsigned int JSHash(const std::string& str)
{
   unsigned int hash = 1315423911;

   for(std::size_t i = 0; i < str.length(); i++)
   {
      hash ^= ((hash << 5) + str[i] + (hash >> 2));
   }

   return hash;
}
```

UNIX object file hash (ELF)

```
unsigned int ELFHash(const std::string& str)
{
   unsigned int hash = 0;
   unsigned int x    = 0;

   for(std::size_t i = 0; i < str.length(); i++)
   {
      hash = (hash << 4) + str[i];
      if((x = hash & 0xF0000000L) != 0)
      {
      if((x = hash & 0xF0000000L) != 0)
      {

      }
      hash &= ~x;
   }
   return hash;
```

Some well known hash functions

Donald E. Knuth in The Art Of Computer Programming Volume 3

```
unsigned int DEKHash(const std::string& str)
{
    unsigned int hash = static_cast<unsigned
  int>(str.length());

    for(std::size_t i = 0; i < str.length(); i++)
    {
       hash = ((hash << 5) ^ (hash >> 27)) ^ str[i];
    }

    return hash;
}
```

Advantages/Disadvantages of hashing

Advantages: (good hash function, not close to full)

- insert is O(1)
- retrieve is O(1)
- delete is O(1)

Disadvantages:

- traversals <u>in order by key</u> is (very) slow
- selection <u>in a range of keys</u> is (very) slow

Next Actions and Reminders

Read CH pp. 567- 591 and pp. 592-598 on Red-Black Trees.

Program 5 is due 12/11, ***if*** you have late days you can use them.