Overview

- Hash Functions
- Java's Builtin Hash Tables

Hash Functions: The Multiplication Method

This method has two steps

1. Multiply the key k by a constant A in the range $0 < A < 1$ and extract the fractional part of kA.

2. Multiply this value by m and take the floor of the result.

In short, the hash function is

$$h(k) = \lfloor m(kA \; mod \; 1) \rfloor \tag{1}$$

where "$kA \; mod \; 1$" means the fractional part of kA, i.e., $kA - \lfloor kA \rfloor$.

Hash Functions: The Multiplication Method (cont.)

An advantage of this method is that the value of m is not critical. It is typically chosen to be a power of 2—$m = 2^p$ for some integer p—since we can easily implement this function:

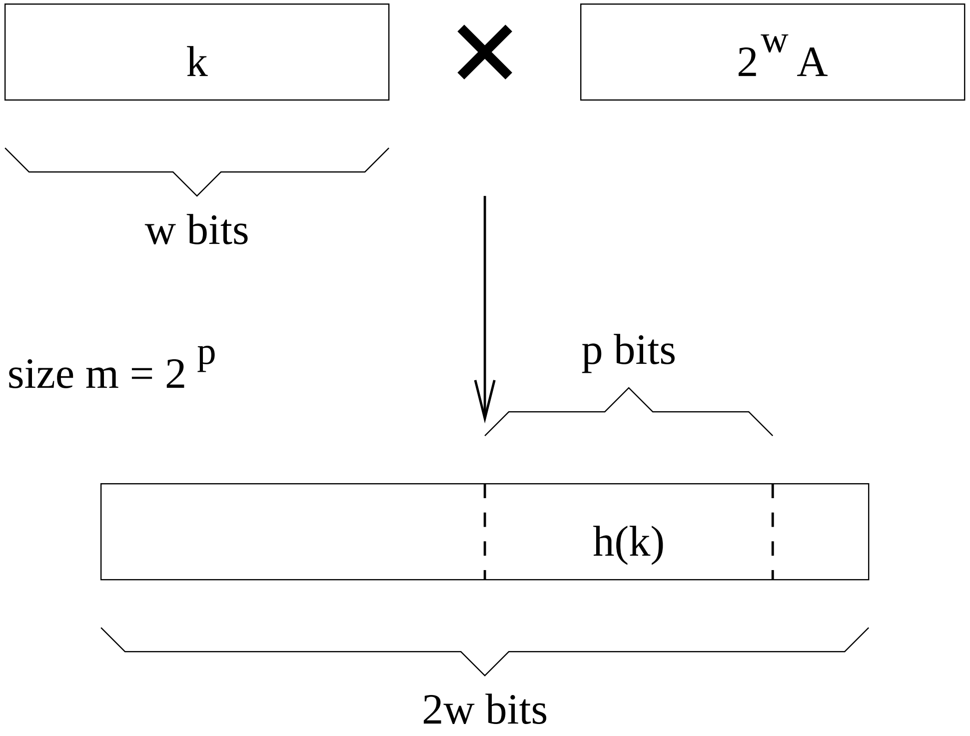
Suppose that the word size of the machine is w bits and that k fits into a word. We first multiply k by the w-bit integer $\lfloor 2^w A \rfloor$. The result is a $2w$-bit value $r_1 2^w + r_0$, where r_1 is the high-order word of the product and r_0 is the low-order word of the product. The desired p-bit hash value consists of the p most significant bits of r_0.

Hash Functions: The Multiplication Method (cont.)

Java "int" $w=32$ $0<A<1$

Java "long" $w=64$

Hash table size $m = 2^p$



Hash Functions: Cyclic Redundancy Check (CRC)

- Let $k[n]$ denote the n^{th} bit of the key.
- Divide the corresponding polynomial $k[n] \times x^n + k[n-1] \times x^{n-1} + ... + k[0]$ by a "magic" polynomial.
  - Note: the coefficients of both polynomials are either 0 or 1.
  - Uses "polynomial arithmetic mod 2": all coefficients are calculated mod 2. The "magic" (divisor) polynomial is commonly called the "generator" polynomial.
- Use the remainder as the hash code.

Hash Functions: CRC (cont'd)

- The CRC algorithm is widely used for detection of errors in data storage and transmission by unreliable media.
  - With a well-chosen generator polynomial, it's possible to detect ALL:
    - single-bit errors
    - two-bit errors
    - errors where an odd number of bits are affected
    - "burst" errors (up to the degree of the generator polynomial)
- The CRC algorithm makes a good hash function.
  - A single-bit difference between two keys yields large differences between the hash codes.
  - Easy to apply to any size key.
    - The implementation can be reduced to repeated table lookup. (The table is dependent on the generator polynomial.)

Java's Builtin Hash Tables

Java's `java.util` package contains a hash table class. This `Hashtable` class extends the `Dictionary` (abstract) class.

A `Dictionary` is an object that associates elements with keys. Every key and every element is an object. In any one `Dictionary`, every key is associated at most one element. Given a `Dictionary` and a key, the associated element can be looked up.

Java's `Dictionary` Class

The `Dictionary` class is defined as follows.

```
public abstract class Dictionary {
    abstract public int size();
    abstract public boolean isEmpty();
    abstract public Object get(Object key)
            throws NullPointerException;
    abstract public Object put(Object key,
                               Object element)
            throws NullPointerException;
    abstract public Object remove(Object key)
            throws NullPointerException;
    abstract public Enumeration keys();
    abstract public Enumeration elements();
}
```

If the key (or element passed to put) is null, a NullPointerException is thrown. If the key passed to get is not found, null is returned.

Java's `Hashtable` Class

The `Hashtable` class is defined as follows.

```
public class Hashtable extends Dictionary
                    implements Cloneable {
    public Hashtable(int initialCapacity,
                     float loadFactor);
    public Hashtable(int initialCapacity);
    public Hashtable(); /* load 0.75 */
    public String toString();
    public Object clone();
    public int size();
    public boolean isEmpty();
    public Object get(Object key)
            throws NullPointerException;
    public Object put(Object key, Object value)
            throws NullPointerException;
    public Object remove(Object key)
            throws NullPointerException;
    public Enumeration keys();
```

Java's `Hashtable` Class (cont.)

```
    public Enumeration elements();
    public boolean contains(Object value);
    public boolean containsKey(Object key);
    protected void rehash();
    public void clear();
}
```

A `Hashtable` has two parameters that affect its efficiency: its capacity and its load factor. The load factor should be between 0.0 and 1.0. When the number of entries in the hash table exceeds the product of the load factor and the current capacity, the capacity is increased, using the rehash method. Larger load factors use memory more efficiently at the expense of larger expected time per lookup. If many entries are to be made in a hash table, creating it with a sufficiently large capacity may allow the entries to be inserted more efficiently than letting it perform automatic rehashing as needed to grow the table.

Java's `Object` Class's `hashCode` Method

Java's `Object` Class defines a `hashCode` Method.

```
public class Object {
    public final Class getClass();
    public String toString();
    public boolean equals(Object obj);
    public int hashCode();
    ...
```

This method is supported principally for the benefit of hash tables such as those provided by the Java library class java.util.Hashtable.