Overview

- Hash Functions
- Hash Tables
computed from the key.

Instead of using the key as an array index directly, the array index is

size proportional to the number of keys actually stored.

directly addressing an array, since a hash table typically uses an array of
number of possible keys, hash tables become an effective alternative to

When the number of keys actually stored is small relative to the total

A hash table is a generalization of an ordinary array.

Hash Tables
(I) \[ \{ I - m \cdots - 0 \} \leftarrow \bigcup \{ h \} \]

- \( h \) maps the universe of keys into the slots of a hash table \( L \) [I - m \cdots - 0].

- The function \( h \) is used to compute the slot from the key \( k \).

- With hashing, an element with key \( k \) is stored in slot \( h(k) \); i.e., a hash.

Hash Tables (cont.)
To cope with collisions, the simplest method is chaining.

- To avoid collisions, \( h \) should appear "random", i.e., adjacent keys should not hash to adjacent slots.

Because \( |\Omega| < m \), collisions are unavoidable. Two keys may hash to the same slot. This is called a collision.

The Problem: Collisions
Removal is identical to lookup.

Length of the list:

Worst case running time is proportional to the corresponding key. So, the worst case running time is $O(1)$.

To look up an element, we search the list belonging to the slot for the element. We simply put it at the front of the list. So the worst case running time is $O(1)$.

To insert an element, we simply put it at the front of the list.

In chaining, we put all the elements that hash to $j$ in the same slot in a linked list. If there are no such elements, slot $j$ contains an empty list.

In chaining, we put all the elements that hash to $j$ in the same slot in a linked list. If there are no such elements, slot $j$ contains an empty list.

Chaining
time. Then the expected case search time is $O(1 + \alpha)$.

We will assume that (1) any given element is equally likely to hash into any of the $m$ slots and (2) the hash value can be computed in $O(1)$ time.

We will distribute the set of $n$ keys to be stored among the $m$ slots on average. The expected case behavior depends on how well the hash function distributes the keys.

- The expected case behavior depends on how well the hash function distributes the keys.
- The worst case behavior of hashing with chaining is $O(n)$ when all keys hash to the same slot, creating a list of length $n$.

Load factor $\alpha$ is defined as $n/m$, i.e., the average number of elements in a chain. Given a hash table with $m$ slots that stores $n$ elements, we define the load factor $\alpha = n/m$.
Thus, searching takes constant time on average.

\[ O = \frac{m}{u} \]

\( \alpha \) and conversely, \( O = n \). If the number of hash table slots is at least proportional to the number of elements in the table, we have

\[ O = m \]

Performance (cont.)
number.

simple method for interpreting each key as a (possibly large) natural

It is usually straightforward in any given application to devise some such

number, Pr becomes \( (112 \times 128) + 116 = 14452 \).

ASCII character set, which has 128 characters. Expressed as a radix-128

bcause \( p = 112 \) and \( t = 116 \) in the

pair of natural numbers \( (112, 116) \) because the identitier \( Pr \) might be interpreted as the

a radix notation. Thus, the identitier \( Pr \) might be interpreted as a natural number expressed in

a character string can be interpreted as a natural number expressed in

a way that is not natural numbers, a way

Most hash functions assume that the universe of keys is the set of

Hash Functions
for \( m \) are primes not too close to powers of \( 2 \). We usually avoid certain values of \( m \), such as powers of \( 2 \). Good values

\[
\text{mod } m
\]

(2)

\[
\varphi(k) = k \text{ mod } m
\]

of \( k \) divided by \( m \), i.e.,

This method maps a key \( k \) into one of \( m \) slots by taking the remainder

The Division Method
where \( n_A \) means the fractional part of \( n_A \), i.e., \( n_A - \lfloor n_A \rfloor \).

\[
\lfloor (n_A \cdot y) \mod m \rfloor = (e(y) \mod m) \cdot (n_A \cdot y)
\]

In short, the hash function is:

1. Multiply the key \( k \) by a constant \( A \) in the range 0 \( < A < 1 \) and
2. Extract the fractional part of \( A \).

This method has two steps:

The Multiplication Method
consists of the most significant bits of \( t \):

and \( t \) is the low-order word of the product. The desired \( p \)-bit hash value

is the high-order word of the product where \( t \) is the \( m \)-bit value \( \sum_{i=0}^{m} t_i z^i \). The result is a word. We first multiply \( k \) by the \( m \)-bit integer \( \sum_{i=0}^{m} k_i z^i \). Suppose that the word size of the machine is \( m \) bits and that \( k \) fits into

we can easily implement this function:

An advantage of this method is that the value of \( m \) is not critical. It is

The Multiplication Method (cont.)
The Multiplication Method (cont.)

Java "long\" w=64
Java "int\" w=32

w bits

p bits

2^p

Hash table size m = 2^p

2^w \times a

w bits

h(k)

\hspace{1cm}

2^w \times a

0 < a < 1

Java "int\" w=32
Java "long\" w=64

Java "int\" w=32
Java "long\" w=64