Hash tables

Goldsmiths Computing

Motivation

A different way to implement a collection, with different performance implications

Definition

A hash table is a data structure that can represent a set, or more generally a map of keys to values (an associative array), by computing a numeric value for each key using a hash function and then using that numeric value to compute an index into an array to look up the value.

Set operations

insert[o]	insert the object o into the set
find[o]	is the object o in the set?

and also

delete[o]	delete the object o from the set

Sets of small integers

Represent sets of non-negative integers smaller than N using an array of size N. e.g. for domain [0,5]:

✓	✗	✗	✓	✗	✓

represents the set $\{0, 3, 5\}$

insert[o] S[o] ← true

find[o] **return** S[o]

delete[o] S[o] ← false

Sets of unbounded integers

Apply the same representation?

✓	✗	✗	✓	✗	✓	⋯									

2^{32} integers? 2^{29} bytes of RAM (512MB)

Sets of unbounded integers

If the expected size of the sets is small (even if the range of possible values is large):

1. choose a reasonable size for the array, say twice expected size
2. reduce the integer to within the range of array indices using a function f(n)
3. store the (unreduced) integer in the array slot

Then

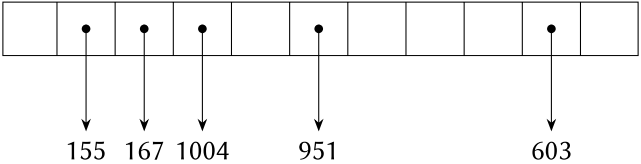
$$\begin{aligned} \text{insert}[o] \quad & S[f(o)] \leftarrow o \\ \text{find}[o] \quad & \textbf{return } S[f(o)] = o \\ \text{delete}[o] \quad & S[f(o)] \leftarrow \text{NIL} \end{aligned}$$

Example

range $[0, 2^{10})$

set size 5

Choose array size of (say) 11 and compute index as f(n) = n mod 11



represents the set {155, 167, 603, 951, 1004}

Complexity analysis

Provided the reducing function f(n) is $\Theta(1)$

insert

$\Theta(1)$ reduction and $\Theta(1)$ memory operations

$$\Longrightarrow \Theta(1)$$

find

$\Theta(1)$ reduction and $\Theta(1)$ memory operations

$$\Longrightarrow \Theta(1)$$

delete

$\Theta(1)$ reduction and $\Theta(1)$ memory operations

$$\Longrightarrow \Theta(1)$$

So what am I not telling you?

Sets of arbitrary things

- compute an integer (a *hash code*) for the things using a *hash function*
  - equal things must have equal hash codes
  - unequal things should be unlikely to share hash codes

computing an integer for the things:

Java `public int hashcode()`

C++ `operator()` functor second template argument to container

equal things must have equal integer codes:

Java `public boolean equals(Object o)`

C++ `operator()` functor third template argument to container

Work

1. Reading
    - CLRS, sections 11.1 and 11.2
    - DPV, section 1.5
    - Drozdek, sections 10.1