Hash tables

Faster search

For fixed data in an ordered array, the binary search algorithm gives O(log(n)) search

For data which varies, a binary tree gives O(log(n)) search, provided it is kept balanced

Can we search any faster?

Yes!

We can use hash tables

The hash table idea

Suppose we have a set of words, a dictionary, and we want search repeatedly to see if a word is included

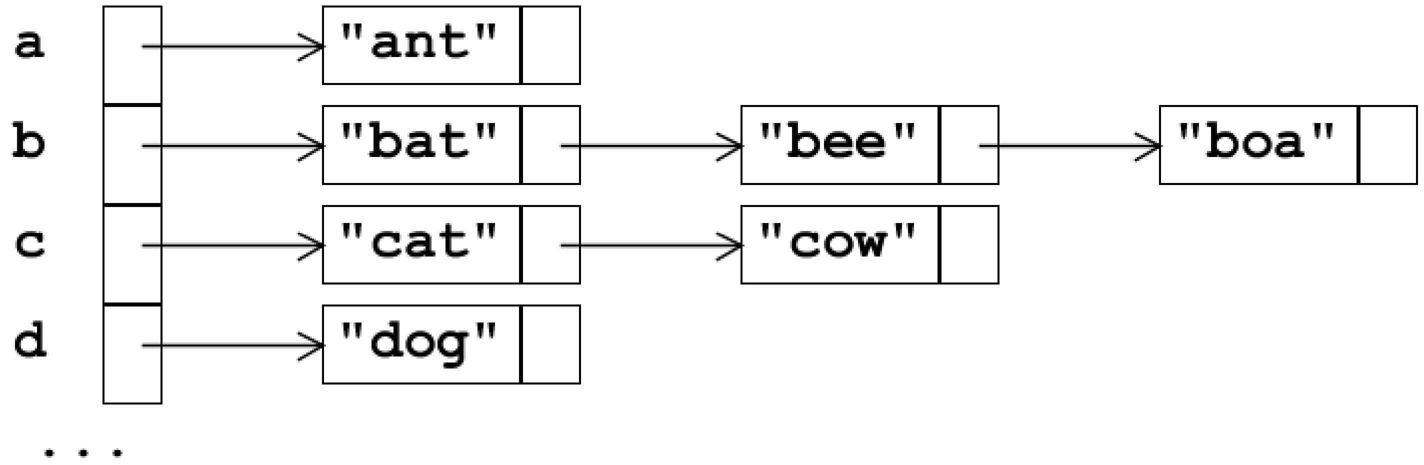
And let's suppose the words contain only lower case letters

We can speed up search by using an array with 26 entries, one for the words starting with each letter

Suppose each array entry contains a linked list

Hash table picture

Here is a 26-entry hash table for three-letter animal names:



On the left is an array of pointers, pictured vertically, indexed by `0` up to `25` by subtracting `'a'` from the first letter

Problems

This hash table could reduce the lookup time to `n/26`, but there are two problems

First, if the words are not evenly distributed between the 26 slots (about the same number in each), then the search time will be worse than `n/26`

Second, `n/26` is still O(n), and we want less than O(n), in fact less that O(log(n))

Hash functions

A function which calculates a number from a string (or other data) is called a ***hash function***

We have been using `h = s[0] - 'a'`

We want (a) the number to be pseudo-random, so that the words generate a good spread of numbers (b) all of the letters to contribute (c) the numbers to be ints, so that large hash tables can be used

A hash function

As with pseudo-random number generators, you should be careful not to invent your own bad one

The first one in the Java language used only a few characters at the start and end of a string, which made it bad for URLs (lots of URLs got the same number, e.g. `http://www.../index.html`)

Now the Java hash function for strings is `for (..i..) h = 31*h + s[i]` ignoring overflow (and it is cached for efficiency)

Hash table size

To reduce search time below O(n), you need a hash table size which is roughly equal to the number of items to be stored

One way is to choose something big and hope the speedup is enough

Another way is to estimate the amount of data in advance

Another way is to make the array dynamic, rehashing all the items into the larger table each time you increase the size

Hash table indexing

A number from a hash function has to be turned into an index, usually just by forming `h % size` to give a number from `0` to `size-1`

Most hash functions have a weakness – the least significant bits are not very random

So one (old) approach is to choose a table size which is a prime, so that `h % size` is still well spread

Better is to design a hash function where the least significant bits are more random, so that arbitrary hash table sizes can be used