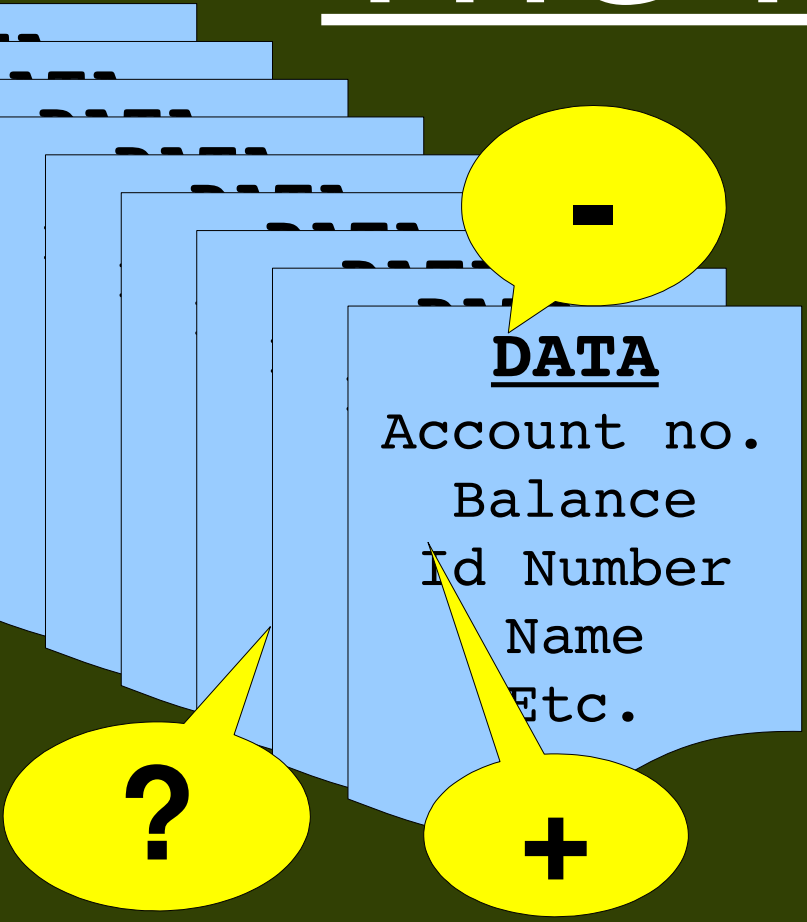


The Problem



DATA
Account no.
Balance
Id Number
Name
Etc.

- Store data
- EG Bank Accounts
 - Account number, Details And the balance
 - 5000+ Accounts
- thousands of additions, deletions and queries per second

Specifics

- add entries
- Lookup entrie's info
- delete and modify entries
- do all of that VERY quickly
- space isn't too much of a problem
- All entries must have a key, EG Acount No.
- Key used to specify which entry must be Queried/modified/deleted

Possible Solutions

- Array

- -Not Dynamic, Deletions and Additions Impractical

- Linked List

- -too slow in lookup/change/delete

- Trees

- -Faster, but still too slow

- The Ultimate Solution: Hash Tables!!!

Hash Tables

Hash tables = constant time

Adding/deleting/querying/modifying in $O(1)$ time!

Actual efficiency depends on:

- Quality of Hash Function
- Memory available
- Input

Bad hash function/Evil input/zero memory
linear time

How they Work

Array of size x

Hash function, $f()$, accepts whatever type the key is

$$0 \leq f(\text{key}) < x$$

$f(\text{key})$ always the same

Entry is stored at position $f(\text{key})$ in the array

Collisions: two or more keys produce same result

There are several ways to handle collisions

The Hash Function

A well chosen hash function is vitally important

Designed for intended data

Produce an even spread, with minimal collisions

Speed & Simplicity (SHA-1 cryptographic hash function will not do!)

Good example function:

<This space intentionally left blank so that
coaches can provide their valuable input :>

Storing Data

Seperate chaining – attach linked lists

Open adressing – Store data in table

Collision Handling

In separate chaining – not a problem, an element is simply appended to the end of the list

In open addressing, probing is required

Probing = looking for somewhere else to put it

Linear probing

Quadratic probing

Double hashing

Linear Probing

When storing:

1. If original spot is available, store there
2. If not, check if the next position is open
If it is store there.
3. If not, check the next one, etc.

When retrieving:

1. Check original spot, if taken but not correct
2. Check the next spot.
3. Continue searching till item or empty space

Clustering can happen which results in reduced efficiency

Quadratic Probing

Similar to linear, except that the interval between probes increases

This helps to alleviate clustering

Double Hashing

The interval between probing is calculated by a second, different hash function

Clustering only occurs when the table is very close to full

Deleting

With separate chaining: this is trivial

With open addressing: its more complicated...

Remember, search stops when it finds empty entry

Potentially separate data from it's intended position

One solution is to use a "place holder"

Can be written over when adding data

Will be treated as a incorrect match and not a empty space when searching.

Resizing

If you know how much data before hand, not really an issue

Seperate chaining: resizing recommended

Open addressing: resizing crucial

As the table's load passes 80%, open addressing becomes very inefficient very quickly

It is recommended to do a resize operation at about 75% load.

Resizing is costly so increase by a ratio
ie. Double the size, don't just add 10 entries

To chain or not to chain

Clearly separate chaining is easiest to implement
No probing, deleting easy, resizing less crucial

Separate chaining also conserves space when load
is low or when data is large

Use separate chaining whenever the data is more
than 4 words