Hash Tables (Key, Content)

- Search based on Key; arbitrary elements
- Insert/Remove with given Key

Eg. McGill ID -> Key; student info -> Content
    username -> Key; transactions, activity -> Content

Lists: Search, Remove O(n), Insert O(1)

Arrays: Sorted -> Search O(log n); Insert, Remove O(n)

Binary Search Trees -> O(height)
  (Heaps only allow retrieval of "special" key)

log n \& m (depending on balancing effort)

Most of the time we can do things in \( \leq O(1) \) (ISp Search)

1ID \rightarrow 9 digit integer \quad a[id] \quad (a holds content)

\uparrow all operations go directly to right location

Mem big!! \quad O(1)!

10^3 array (much bigger than array actually needed for
"active" data)

\( \approx \) amount of active data or
a cost fraction

"Bucket" of data (hold more than 1 element)
  (e.g. Linked List, Array, Bin. Search Tree...)
Key → index in the array (choose which bucket)

Hash Function

Size of array in B \{ \rightarrow \text{Key mod B valid index} \\
\text{Key integer} \}

"Java: Key \% B"

Design a good hash function is crucial

Evenly spread elements \{ items give \( m/B \) items in each bucket \}

if \( B \) is a fraction of \( m \) \( \Rightarrow \) bucket would be constant size

Elements mapped to just a few buckets \( \rightarrow \) Bucket \( B \)

\[
\begin{array}{c}
0 \\
1 \\
\vdots \\
K-1 \\
0 \\
B-1
\end{array}
\]

\( O(m) \) (bad efficiency)

\( O(1) \) (bad distribution)

McGuff ID's

\[ 260 \text{ ID} \rightarrow 10k \text{ away} \]

index maps lots of users to one of my few buckets (bad!)

Domain knowledge comes to play
Integers as key

what B (size) should be hash for

( Guarding against "unlucky" distributions )

E.g., many keys are a multiple of some int (say 5)

Taking mod (table of size multiple of 5)

\[
\begin{array}{c|c|c|c|c|c}
0 & 5 & 10 & \ldots \\
\hline
{\text{empty!}}
\end{array}
\]

Choose B to be prime (fairly large), e.g. 101

E.g., keys 155 255 355

\[\text{mod 101} \]

54 53 52

155 255 355

\[\text{mod 100} \]

\[55 \text{ (same bin!)}\]
B = 101; all keys are multiples of 101 still bad

Why are prime hash table sizes good?
K \rightarrow \text{divisible by some prime factor } p \Rightarrow K = p \times m
B \rightarrow \text{also divisible by } p \Rightarrow B = p \times l

K \mod B = (p \times m - p \times l \times m) = p(m - l \times m)

\downarrow \text{result of the division}

\downarrow \text{multiple of } p!

Bins that are multiples of p are "lucky!"
Happens for all divisors of B!
B \text{ composite} \Rightarrow \text{lots of "unlucky" cases!}

Java: HashMap class

\rightarrow \text{size}

\rightarrow \text{load } \epsilon(0, 1)

\text{If load specified is exceeded} \rightarrow \text{internally away in rehash}

\text{If you specify load} \rightarrow \text{hardly close to 1 (default 0.75)}
Memory vs computation time:
Small table \( \rightarrow \) load \( \rightarrow \) running time dominated by access into bucket

Large table \( \rightarrow \) load \( \rightarrow \) running time approaches \( O(1) \) on avg.

Object class has a hashCode() method

\( \text{Object1.equals(Object2)} \Rightarrow \text{Object1.hashCode}() == \text{Object2.hashCode}() \)

- Has strings \( \rightarrow \) ASCII code of characters
  
  \( \text{process it, e.g., sum} \)

  \( \text{ux mod } 75 \)

HashMap \( \rightarrow \) "buckets" can be any collection (including HashMap!)