Today

- Hash tables (cont)

Assignments

- HW7 out (due Tues)
- HW8 out soon
- Quiz 8 out today (due Tues)
Hash Tables

The basic idea:

- Keep an array of elements ("buckets")
- Define a "hash" function $h : \text{Key} \rightarrow \text{Index}$
- Make $h$ fast relative to $n$ (i.e., $h$ should be constant time relative to $n$)
“Perfect Hash Functions”

- Map each key to a unique array index ... i.e., no to keys map to same index
- Hard if you do not know all key values to expect
- You may also have more keys than array indexes

Most Hash Functions

- Map two or more keys to the same index
- If this happens, creates a “collision”
- We have to have ways to deal with collisions (more later)
- ... but we also want hash functions that minimize collisions
Examples of Hash Functions

Assumptions

- keys are positive integers (for now)
- we have a hash table with 100 (0 to 99) slots / indices

Approach 1: “Selecting Digits”

- Select specific digits of the key to use as the hash value

- Lets say keys are 9-digit employee numbers
  - $h(k) = 4$th and 9th digit of $k$
  - For example: $h(001364825) = 35$

- Here we insert (find) entry with key 001364825 at table[35]

- This is a fast and simple approach, but

- May not evenly distribute data Q: ... why not?
Approach 2: “Folding”

- Add (sum) digits in the key

- Lets say keys are again 9-digit employee numbers
  \[ h(k) = i_1 + i_2 + \cdots + i_9 \]
  where \( h(k) = i = i_1i_2 \ldots i_9 \)

  - For example: \( h(001364825) = 29 \)
  - Store (retrieve) entry with key 001364825 at \texttt{table}[29]

- This is also fast, but also may not evenly distribute data

- In this example, only hits ranges from 0 to 81

- Can pick different (similar) schemes ... e.g., \( i_1i_2i_3 + i_4i_5i_6 + i_7i_8i_9 \)
**Approach 3: Modular Arithmetic (via Modulo Operator)**

- Sometimes we end up with indexes **outside** of the range of table indexes

- Can use the modulo operator `%` to map values to valid table indexes

  - \( h(k) = f(k) \mod \text{table size} \)  
    ... for \( f : \text{Key} \rightarrow \text{int} \)

- Note the modulo operator \( a \mod n \) returns the “remainder” \( r \) of \( a \div n \)

  \[
  r = a - n \left\lfloor \frac{a}{n} \right\rfloor 
  \]

Examples:

\[
\begin{align*}
5 \mod 2 &= 5 - 2 \left\lfloor \frac{5}{2} \right\rfloor = 5 - 2 \cdot 2 = 1 \\
7 \mod 4 &= 7 - 4 \left\lfloor \frac{7}{4} \right\rfloor = 7 - 4 \cdot 1 = 3 \\
36 \mod 12 &= 36 - 12 \left\lfloor \frac{36}{12} \right\rfloor = 36 - 12 \cdot 3 = 0 \\
36 \mod 13 &= 36 - 13 \left\lfloor \frac{36}{13} \right\rfloor = 36 - 13 \cdot 2 = 10 \\
4 \mod 7 &= 4 - 7 \left\lfloor \frac{4}{7} \right\rfloor = 4 - 7 \cdot 0 = 4
\end{align*}
\]

- In our running example we can use the key and table size directly ...

  \( h(001364825) = 1,364,825 \mod 100 = 25 \)

- Key values used may require carefully chosen table sizes

  - Since keys may not be random (e.g., multiples of 2 or 10 or ...)
  - E.g., 110 mod 100, 210 mod 100, 310 mod 100, etc
  - One approach is to use prime numbers (to help distribute values evenly)
  - Like 101 in this case