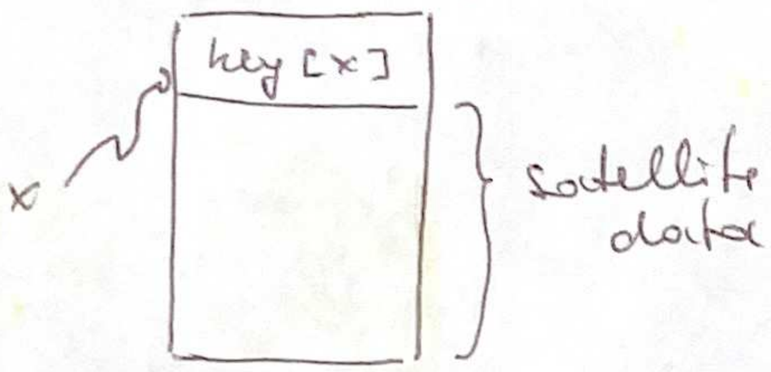
Hashing, Hash Functions

Symbol-table problem (compilers)

Table S holding records



record

Operations:
- insert (S, x): $S \leftarrow S \cup \{x\}$
- delete (S, x): $S \leftarrow S - \{x\}$
- search (S, k): return x ■ s.t. key[x] = k or nil if no such x

} dynamic set

state set is only lookup, etc...

Direct access table

Suppose keys are drawn from $u = \{0, 1, \ldots, m-1\}$
Assume keys are distinct
Set up array $T[0 \ldots m-1]$ to represent dyn. set S

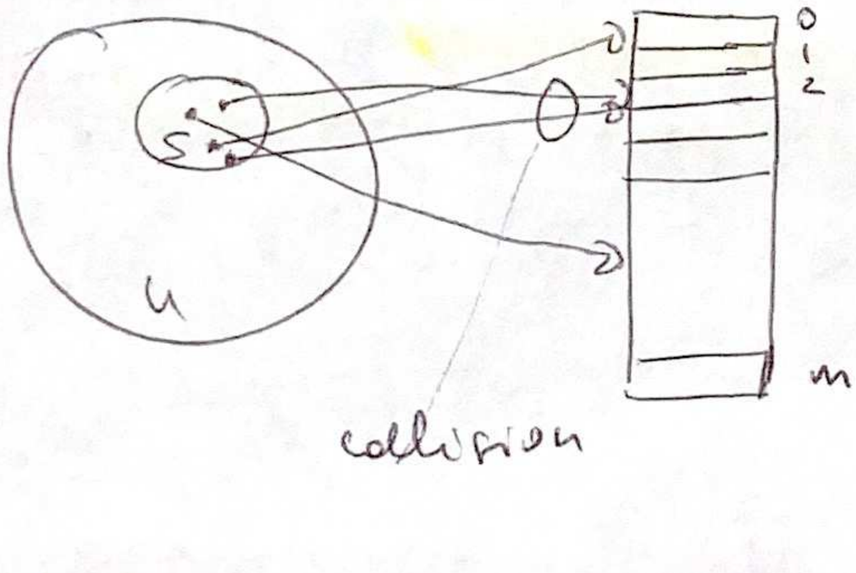
$$T[k] = \begin{cases} x \text{ if } x \in S \text{ and } key[x] = k \\ nil \text{ otherwise} \end{cases}$$

Ops take $\Theta(1)$ time

e.g. 64 bit #s drawn, need a table of size 2^{64}
↳ need hashing

Hashing

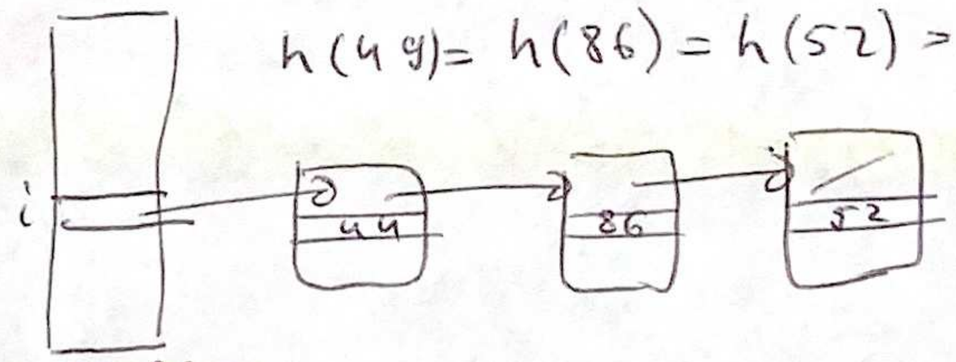
Hash function h maps keys "randomly" into slots of table T.



when a record to be inserted maps to an already occupied slot, a collision occurs.

Resolving collisions by chaining

idea: link records in same slot into list.



Analysis

Worst case: every (valid) key hashes to the same slot. → long linked list
access takes $\Theta(n)$ time if $|S| = n$

Average case

assumption of simple uniform hashing
- each key is equally likely to be hashed to any slot in T, independent where other keys are hashed

Def: the load factor of a hash table with n keys and m slots is $\alpha = n/m$ = ■ ave # keys per slot. ← expectation of binomial

Expected unsuccessful search time: $\Theta(1+\alpha)$

↑ cost of hash & access to slot ↑ cost of lost search

↗ also for successful search $\Theta(1+\alpha)$

Expected search time = $\Theta(1)$

if $\alpha = O(1)$, i.e. $n = O(m)$

↑ if n grows m has to grow

Represent a dynamic set with cost ■ time ops as long as:
- $n = O(m)$
- simple uniform hashing

Choosing a hash function:

- Should distribute keys uniformly into slots
- Regularity in key distribution should not affect uniformity

Division method

$h(k) = k \bmod m$ ← does not have to be eq. to size of table can be around the size (len)

- don't pick m with small divisor d

Ex. $d = 2$ and all keys even ← even # mod even # is even

=> odd slots never used

Ex $m = 2^r$ => hash does not depend on all bits of k

$k = 10111000111\underbrace{0110}_{h(k)}10$ ← 2^6 $r = 6$ $m = 2^6$

h does not depend on other bits

Pick m = prime not too close to power of 2 or 10 } good heuristic

common bases with regularities in the world

However, the division method is compute-intensive.

Multiplication method

number of slots $m = 2^r$, computer has w-bit words ← Bits within a word after multiplication

$h(k) = (A \cdot k \bmod 2^w) \text{ rsh } (w-r)$

↑ fixed odd integer $2^{w-1} < A < 2^w$

↑ right shifted by $\cdot$ bitwise

- Don't pick A too close to 2^{w-1} or 2^w
- Fast method: mult. mod 2^w faster than division
  rsh is fast

Ex $n = 8 = 2^3$, $w = 7$ ← full word

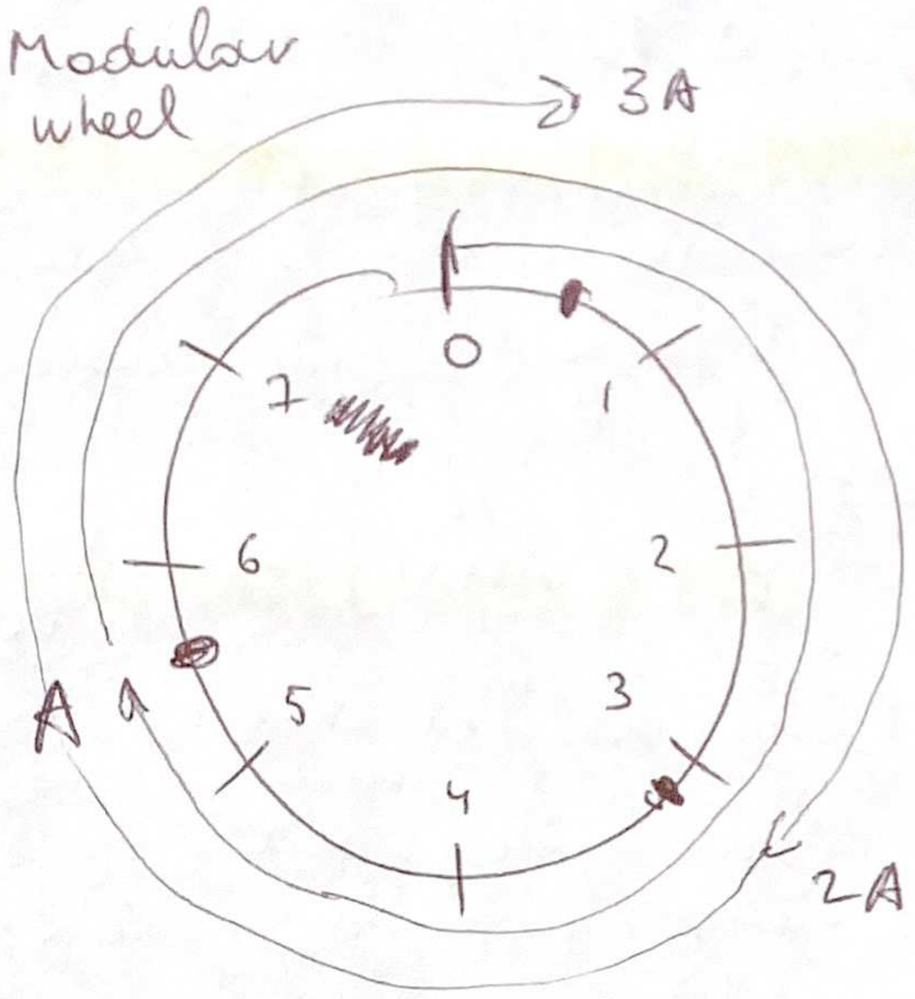
```
  1011001   = A
  1101011   = k
-----------
1001010011 0011
```

high-order bits ignored with mod 2^w ↑ two-word product

```
011 0011
```

$h(k)$ → rsh removes 4 bits

Modular wheel



k spins the wheel... → heuristic

Resolving collisions by open addressing

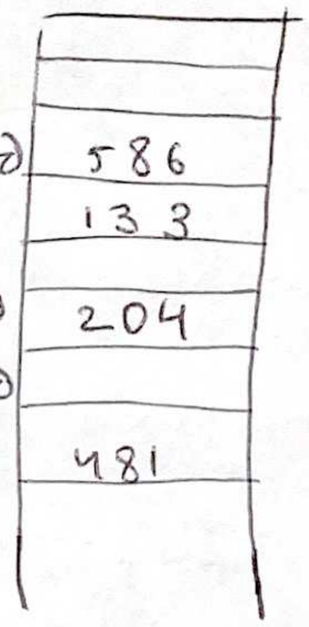
- No storage for links
- Probe table systematically until an empty slot is found (hash function after hash function)
- $h: U \times \{0, 1, \ldots, m-1\} \to \{0, 1, \ldots, m-1\}$

  U = universe of keys; probe # ∈ index of a hash function

- Probe seq. should be permutation ← given a key, the sequence of slots hit, no repeating
- Table may fill up, $n \leq m$
- Deletion is difficult

Ex: Insert $k = 496$

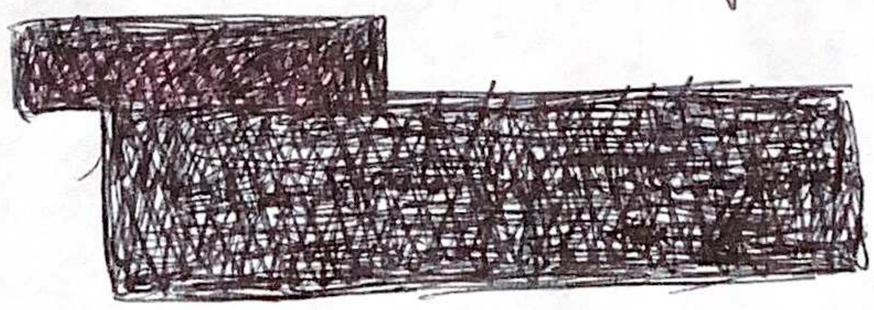
0. Probe $h(496, 0)$
1. Probe $h(496, 1)$
2. Probe $h(496, 2)$



keep track of max part # of insertions, no need to continue attempts if reached

Search - same probe sequence

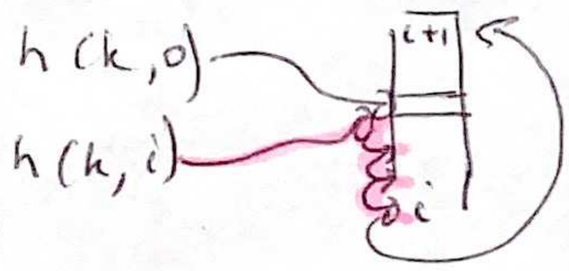
- successful - find record with key
- unsuccessful - find nil



Probing strategies

- Linear - $h(k, i) = (h(k, 0) + i) \bmod m$

  "primary clustering" - long runs of filled slots



- Double hashing - $h(k, i) = (h_1(k) + i \cdot h_2(k)) \bmod m$

  excellent method, usually pick $m = 2^r$ and $h_2(k)$ odd; compute $h_2(k)$ upfront; $m = 2^r$ with multiplication method

Analysis of open addressing

(stronger than simple uniform hashing)

Assumption of uniform hashing:

each key is equally likely to have any one of the $m!$ perms as its probe seq., indep. of other keys

Theorem

$$E[\#\text{probes}] \le \frac{1}{1-\alpha} \quad \text{if } \alpha < 1 \quad (\text{i.e. } n < m)$$

Pf. (unsuccessful search):

1. probe always necessary
   with prob. n/m collision $\Rightarrow$ 2. probe is necessary
2. probe prob. of collision $\frac{n-1}{m-1}$ $\Rightarrow$ 3. probe is necessary
3. probe prob. of collision $\frac{n-2}{m-2}$ ----

Note: $\frac{n-i}{m-i} < \frac{n}{m} = \alpha$ for $i = 1, 2, \dots, n-1$, assumption $n < m$ (necessary for open addressing)

$$E[\#\text{probes}] = 1 + \frac{n}{m}\left(1 + \left(\frac{n-1}{m-1}\right)\left(1 + \frac{n-2}{m-2}\left(\dots\left(1 + \frac{1}{m-n}\right)\right)\dots\right)\right)$$

$$\le 1 + \alpha\left(1 + \alpha\left(1 + \alpha(\dots 1+\alpha)\dots\right)\right)$$

$$\le 1 + \alpha + \alpha^2 + \alpha^3 + \dots$$

$$= \sum_{i=0}^{\infty} \alpha^i = \frac{1}{1-\alpha}$$

$1 + \frac{n}{m} + \frac{n}{m}\left(\frac{n-1}{m-1}\right) + \frac{n}{m}\left(\frac{n-1}{m-1}\right)\left(\frac{n-2}{m-2}\right)\dots$

$\alpha < 1$ const $\Rightarrow O(1)$ probes in expectation

- Table $\alpha = \frac{1}{2}$ 50% full $\Rightarrow \le 2$ probes in expectation
- Table 90% full $\Rightarrow \le 10$ probes in expectation

const goes up
need to keep α low

$\Rightarrow$ keep the table not full