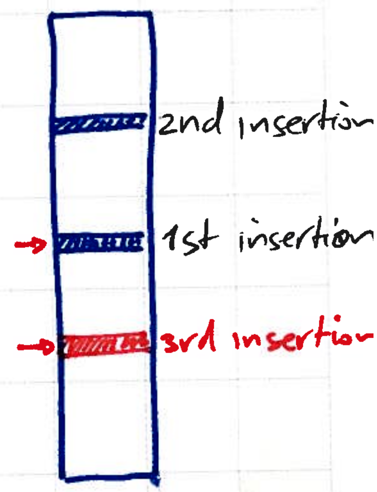
6.006 Lecture 7: Hashing 3

- Open Addressing
- Uniform hashing + analysis
- Advanced topics: universal, perfect, & cryptographic hashing

- CLRS 11.4, 11.3.3, 11.5

Open Adressing



- No linked lists
- Collision? Store elsewhere in hash table
- More collisions? Probe more; may need to probe m-1 times to find an empty slot
- Hash function ~~[illegible]~~ of key k is now a sequence of probes $\langle h(k,0), h(k,1), \ldots h(k,m-1) \rangle$

  sequence must be a permutation of $0, 1, 2, \ldots, m-1$

  a key maps to a permutation

- Clearly, load factor $\alpha \leq 1$.

Insert(k,v):

```
for i in range(m):
    if T[h(k,i)]==None: T[h(k,i)]=(k,v)
                        return
raise Exception('full')
```

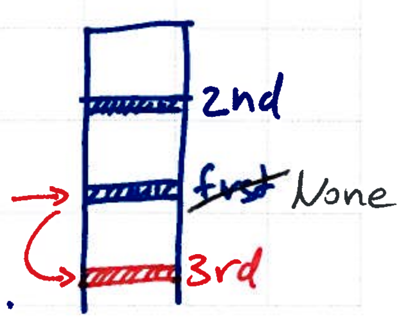
Search(k):

```
for i in range(m)
    if T[h(k,i)]==None: return None #not in table
    if T[h(k,i)][0]==k: return T[h(k,i)]
return None
```

Delete(k):

#tricky! setting T[h(k,i)]=None may cause search to fail (e.g. deleting the first element inserted in the example causes the third not to be found

- Find key
- Replace by 'Deleted'
- Skip ~~oved~~ over 'Deleted' in search but use 'Deleted' slots in insert.



How to construct $h(k,i)$

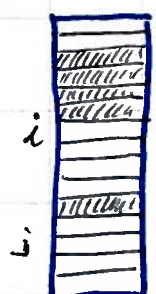
- What do we want?

  for chaining, we want simple uniform hashing: each key is equally likly to hash to any slot.

- For open addressing, we want uniform hashing: each key is equally likly to hash to any of the $m!$ probe sequences (permutations of $0, 1, \cdots m-1$).
- Harder to acheive, but double hashing works well.

Linear probing

- Start with an ordinary hash function $h'(k)$
- $h(k,i) = (h'(k) + i) \bmod m$
- start at $h'(k)$ and scan sequentially
- Not good: only m possible sequences, leads to clustering



which slot is more likly to get filled next? $\underline{i}$, leads to clustering

Double hashing

- $h(k,i) = (h_1(k) + i \cdot h_2(k)) \bmod m$
- to ensure $h(k,*)$ hits all slots, make $h_2(k)$ and m relatively prime. Ex: $m = 2^r$, $h_2(k)$ odd

Expected length of probe sequences (unsuccessful)

Assume uniform hashing

$Pr[\ell \geq 1] = 1$ $\quad$ ℓ is the length of the sequence

$Pr[\ell \geq 2] = 1 \cdot \frac{n}{m} = \alpha$ $\quad$ first slot was occupied

$Pr[\ell \geq 3] = 1 \cdot \frac{n}{m} \cdot \frac{n-1}{m-1} \leq 1 \cdot \frac{n}{m} \cdot \frac{n}{m} = \alpha^2$ $\quad$ relies on $\frac{n-j}{m-j} \leq \frac{n}{m}$ for $n \leq m$, $j \geq 1$

$\vdots$

$E[\text{length of probe sequence}] =$

$$= \sum_{i=1}^{m} i \cdot Pr[\ell = i]$$

$$= \sum_{i=1}^{m} i \left(Pr[\ell \geq i] - Pr[\ell \geq i+1] \right)$$

$$= \left(Pr[\ell \geq 1] - Pr[\ell \geq 2]\right) + 2\left(Pr[\ell \geq 2] - Pr[\ell \geq 3]\right) + 3(\cdots) + \cdots$$

$$= \sum_{i=1}^{m} Pr[\ell \geq i] \leq 1 + \alpha + \alpha^2 + \cdots + \alpha^{m-1} \leq \sum_{i=0}^{\infty} \alpha^i = \frac{1}{1-\alpha}$$

Examples:

$\alpha = \frac{1}{2}$ (table half occupied) $\quad$ two probes expected

$\alpha = 0.9$ $\quad$ ten

$\alpha = 0.99$ $\quad$ a hundred

Open addressing vs	Chaining
cost explodes as α approaches 1	cost rises gently with α
No memory allocation (except to resize), cache efficient	allocates memory as chains grow (constant overhead)
Hard to find a really good h	Easy to find a good h
easier to implement in hardware	

Advanced topics in hashing

• Universal hashing (back to chaining)

For any h there are collisions; if we are unlucky all the keys may hash to 1 slot.

Solution: Don't use a fixed h; choose it at random

Universal hashing: random selection of h should guarantte $\Pr[k_1\ k_2 \text{ collide}] \leq \frac{1}{m}$

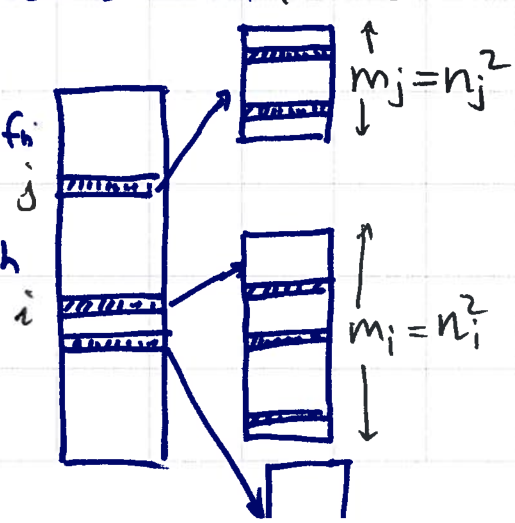
$h(k) = ((ak+b) \bmod p) \bmod m$ (last lecture) works.

random

• Perfect hashing: $\Theta(1)$ worst-case search (fixed set of keys)

- Primary table stores pointers to secondary tables + their hash fn
- Make secondaries large enough so prob. of any collision $\leq \frac{1}{2}$
- Try secondary hashes until no collisions at all

Space is still $\Theta(n)$ (expected)



- **Crypographic Hash Functions** (No table!)

- $h(k)$ such that: (1) given $h(k)$ it is hard to find k' s.t. $h(k') = h(k)$
  (2) it is hard to find k_1, k_2 such that $h(k_1) = h(k_2)$

- Numerous applications! Ex: storing passwords
  Password file contains $h(pass_{sivan})$, $h(pass_{...})$
  to check login attempt: send $h(attempt)$ to server
  Passwords not sent in clear text
  can't steal identity even if pass. file stolen

- Also: digital signatures, proving you have a file without showing it, authenticating files, ...

- Actual widely used crypto hashes:
  MD5 (Prof Rivest), broken!
  SHA-1 broken!

- The race for designing the next secure hash function is on; MD6 ?