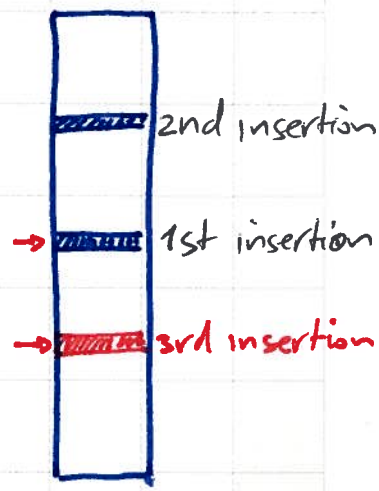
6.006 Lecture 7: Hashing 3

- ☐ Rolling hashes
- ☐ 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 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 $\alpha \leq 1$.

Rabin-Karp String matching & Rolling hashes

Given pattern $P[1..m]$
text $T[1..n]$ } lists of characters

does P occur in T?

E.g. find "ATG" in "AATCGC..."

Idea:

- Compute $h(P)$
- for each length-m window of T, $T[i, \cdots i+m-1]$

$$\underbrace{A\,A\,T}\;C\;G\;C$$
$$\underbrace{A\,A\,T\;C}$$
$$\underbrace{A\,A\,T\;C\;G}$$
$$\ddots$$

compute $h(T[i \cdots i+m-1])$ and compare to $h(P)$

if $=$: check to see if really a match

if $\neq$: move on to next i

- Use a hash function h s.t. can compute $h(T[i \cdots i+m-1])$ from $h(T[i-1 \cdots i+m])$ easily

$\Rightarrow$ rolling hash

Example of a rolling hash

Alphabet $A = 0$, $C = 1$, $G = 2$, $T = 3$

h = value of string % p for prime p

T = C T A T T A C G T

1 3 0 3 3 0 1 2 3 4 ← base 4

1 8 4 0 9 1_{10}

453 mod p = 1009

delete C

T A T T A C G T G

add G

value of leading c = $1 \cdot 4^8 = 1 \cdot 960$ mod p, precompute!

so dropping leading c ⇒ h = 453 − 690

= 502 mod p

shifting and adding a G:

$h = 4 \cdot 502 + 2 = 1001 \bmod p$

shift G

subtract ×1
multiply ×1
add ×1

} cost of moving h to next window

Insert(k,v):

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

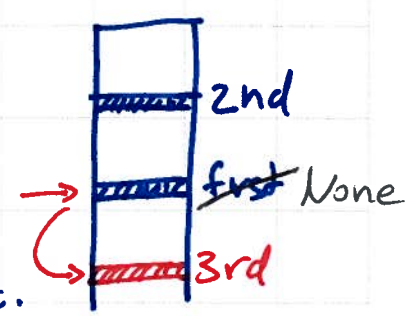
Search(k):

```
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 ~~over~~ 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

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 guarantee $\Pr[k_1\ k_2 \text{ collide}] \le \frac{1}{m}$

$h(k) = ((ak+b) \bmod p) \bmod m$ (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 <u>any</u> collision $\le \frac{1}{2}$
  - Try secondary hashes until <u>no</u> collisions at all

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