Problems: what to do with deletes?

Ex: suppose we delete the key at location 3

if we put NIL at location 3 and later do a search for key[x], we’ll conclude key[x] is not in the table

need to keep a special marker for free slots that used to have something in them

after many insertions and deletions, do not have any idea how long Search takes

ex: we fill the table and then delete most entries but entries left over are the ones whose insertion took a long time; so, even though the table is almost empty, searches take a long time
Any alternatives to chaining as the collision resolution?

Open addressing

all the elements are stored in the table itself, no linked lists
load is always 1

table fills up and cannot take any more elements when n=s
how is it done: compute a sequence of slots for a given key
h(k, 1), h(k, 2), h(k, 3), ....

insert(T, k): successively probe hash table according to the sequence
until we find an empty slot

ex: if the sequence for a key[x] is 4, 1, 11, 3, 9.... and entries
4, 1, 11 and 3 are occupied,
we put key[x] into slot 9

i = 0
repeat j = h(k, i)
    if T[j] = NIL then T[j] = k and return
    else i = i + 1
until i = s
error “table overflow”
Designing a good hash function

Division method: \( h(k) = k \mod s \)

good values for \( s \) are primes not too close to exact powers of 2

Multiplication method: \( h(k) = \lfloor s \cdot (kA \mod 1) \rfloor \)

where \( A \) is a constant between 0 and 1

for example \( A \approx (\sqrt{5} - 1) / 2 \) works well

\( s \) is usually chosen to be \( 2^p \), where \( p \) is a prime number

\( kA \mod 1 \) is simply the fractional part of \( kA \)

problem with both methods: adversary could mess things up bad

Universal hashing: adversary is powerless (just like in Quicksort)

store a large collection of good hash functions and in the beginning of running your program, pick a random one from this collection
Simple uniform hashing
  any given element is equally likely to hash into any of the s slots

Suppose we have n values in a table T of size s
Load=ave number of values per table slot
  $\alpha = \frac{n}{s}$
Assuming simple uniform hashing,
  unsuccessful search takes $\Theta(1+\alpha)$ exp-time
  successful search takes $\Theta(1+\alpha)$ exp-time

Ex: if $n = O(s)$, then $\alpha = O(1)$ and Search takes $O(1)$
Hash Tables:

- element with key k is stored in slot h(k)
- hash table T[0...s-1] and hash function h from U to {0,1,..,s-1}

Problems:

- collision = two keys hashing to the same slot

Solution:

- get a good hash function that acts “random”
  - (but must be deterministic: h(k) should produce one value only)
  - but when |U|>s, collisions are unavoidable
  - still need some way to deal with collisions

Collision resolution by chaining:

- Insert(T,x): insert x at the head of list T[h(key[x])]
- Search(T,k): search for key k in list T[h(k)]
- Delete(T,x): delete x from the list T[h(key[x])]

Running times: assume it takes O(1) time to compute a hash of a key

- Insert: O(1)
- Delete: O(1) if lists are doubly linked
- Search: depends on the length of the list, which depends on hash function
Lecture 8: Hashing

Data structure to support operations: Insert, Search, Delete on elements with keys
Ex: variables in a compiler have variable names (key)

if the key universe $U$ is small: for example $U=\{0,1,2,\ldots,s-1\}$

Direct-address table: make table $T$ of $s$ entries
   element with key $k$ is stored in slot $k$
Search($T,k$): return $T(k)$
Insert($T,x$): $T[\text{key}[x]]=x$
Delete($T,x$): $T[\text{key}[x]]=\text{NIL}$
   Space=$O(s)$
   Each operation takes $O(1)$ time

if the universe $U$ of keys is much larger than the the number of keys ever stored -- can’t do direct-addressing