

---

## “Incremental” Hashing

---

### Reading:

- None in your text.
- Optional reading: ch 12 of *File Structures: An Object-Oriented Approach with C++*, by Folk et al.

157

## Incremental Hashing

### General Approach

As records are inserted, if performance becomes too low, grow the file.

- I.e., “split” one bucket and disperse its records; some stay put and others go to a new bucket.
- This reduces overflow (collisions to full buckets) and hence reduces the # of file accesses during search.

As records are deleted, if space usage becomes too poor, shrink the file.

- I.e., merge two buckets into one.
- This reduces the total # of buckets, and hence reduces waste.

159

## A Problem

Performance degrades if the file becomes heavily loaded,

i.e., if  $\frac{\text{actual-number-of-recs}}{\text{num-buckets} \times \text{bucket-size}}$  gets large.

To make things better, it may be worthwhile to increase the number of buckets (and reorganize the data).

This general idea is called **incremental hashing**.

Guess what? There are many ways to do it.

158

File growth and shrinkage is incremental, i.e.:

- It happens on the fly.  
We do it during insertions and deletions, if needed.
- It happens in small amounts.  
We split one bucket rather than rehashing the whole file.

Possible measures of performance include:

- load factor
- average # of disk accesses per search.

160

## Method I: Linear Hashing

### Method

- When performance becomes too poor, split bucket 0. (Yes, this is arbitrary.)
- Split it by doubling the mod factor and re-hashing its contents. *E.g.*,  
 $h(k) = k \bmod 3$  becomes  
 $h(k) = k \bmod 6$ .
- Next time, split bucket 1, then 2, etc.
- Keep a counter to remember which buckets have been split.  
 Unsplit ones use the old hash function.  
 Split ones use the new.

Merging is analogous but opposite.

	old	new
# buckets	$T : 0 \dots (T - 1)$	$T + 1 : 0 \dots T$
hash fcn	$h(k) = k \bmod 3$ $h(k) = k \bmod T$	$h(k) = k \bmod 6$ $h(k) = k \bmod 2T$

**Guarantee:** Every element of bucket 0 will either stay put, or land in the new bucket  $T$ .

More generally, if we split bucket  $b$ , every record will either stay put, or land in the new bucket  $T + b$ .

Let  $k$  be the record's key.

If it was in bucket  $b$  originally, we know

$$k \bmod T = b.$$

So  $k$  must have been one of these:

$$b \quad T + b \quad 2T + b \quad 3T + b \quad 4T + b \quad 5T + b \quad \dots$$

161

162

So when we hash  $k$  with the new hash function  $h(k) = k \bmod 2T$ , we get either:

- $b$ , in which case the record stays put, or
- $T + b$ , in which case it goes to the new bucket,  $T + b$ .

### Questions

Will linear hashing work if we use open addressing to solve collisions?

Why split the “next” bucket? Why not the culprit, *i.e.*, the one we inserted to when we passed the performance threshold?

Decision: What if the split fails, *i.e.*, everything happens to stay put? We could split again.

What happens when we've split all the original buckets?

163

164

## Method II: Extendible Hashing

Build a dynamic directory (in memory for speed) that copes with the varying load factor.

- Hash function takes you to a *directory* entry, rather than directly to a bucket.
- Because buckets are pointed to, needn't be consecutive in the file. So can add and remove buckets as desired.
- Directory must grow and shrink with number of buckets.
- So # of places to hash to changes. Cope by using only the first so many bits of  $h(\text{key})$ ; change this as necessary to change size of directory.
- If using  $d$  bits, directory size is  $2^d$ .
- So have capacity for  $2^d$  buckets, but can start with fewer; even just one.

165

## How to “grow” the file

When a bucket overflows:

- Split the one bucket in two.
- Half of the directory entries that pointed to the old bucket will still do so, and half will point to the new bucket.

Eventually, we may reach a point where we can't split a bucket this way.

- This occurs when only one directory entry points to the bucket we want to split.
- Then we double the directory size, and re-organize.

166