Hashing

Hash Tables

Hash function h: search key -> [0, ..., B-1].

Buckets are blocks, numbered [0, ..., B-1].

General idea: If a record with search key K exists, then it must be in bucket h(K).

- Cuts search down by a factor of B.
- One disk I/O if there is only one block per bucket.

Hash Table Operations

HashTable Lookup

- For record(s) with search key K, compute h(K); search that bucket.

HashTable Insertion

- Put in bucket h(K) if it fits; otherwise create an overflow block.
- Overflow block(s) are part of bucket.

HashTable Deletion

- Compute h(K); search bucket for record(s) with key K and delete entry

Example with 2 Records/Bucket

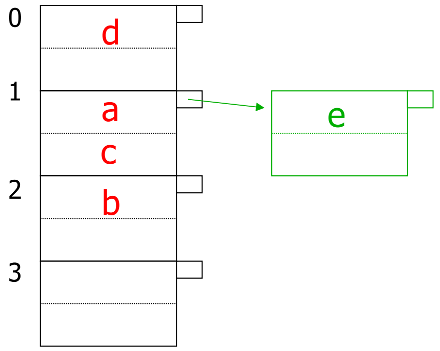
INSERT:

h(a) = 1

h(b) = 2

h(c) = 1

h(d) = 0



h(e) = 1

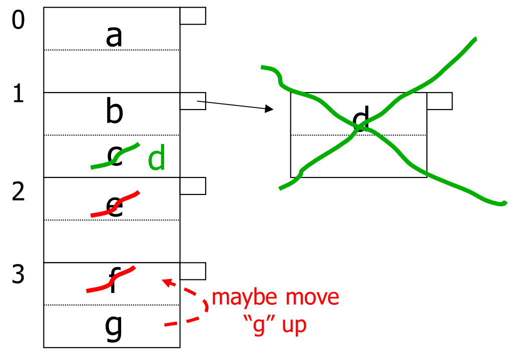
Example: Deletion

Delete:

e

f

c



How full should a Block be?

Try to keep space utilization

between 50% and 80%

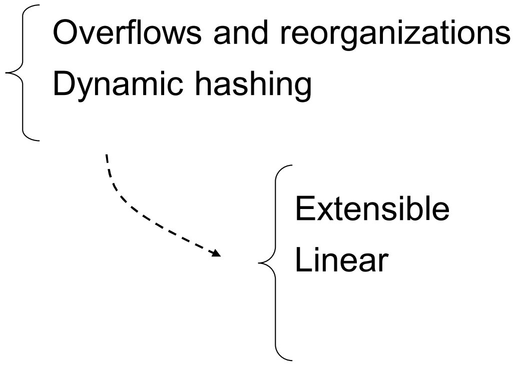
$$\text{Utilization} = \frac{\text{\# keys used}}{\text{total \# keys that fit}}$$

If < 50%, wasting space

If > 80%, overflows significant

depends on how good hash function is and on # keys/bucket

How do we cope with growth?



Overflows and reorganizations

Dynamic hashing

Extensible

Linear

Dynamic Hashing Framework

Hash function h produces a sequence of bits.

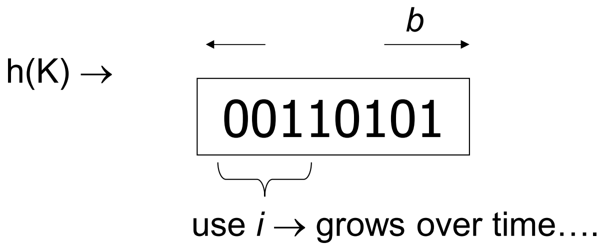
Only some of the bits are used at any time to determine placement of keys in buckets.

Extensible Hashing

- Keep parameter i = number of bits from the beginning of h(K) determine the bucket.
- Bucket array now = pointers to blocks.
- A block can serve as several buckets.
- For each block, a parameter j <= i tells how many bits of h(K) determine membership in the block.
- I.e., a block represents 2^{i-j} buckets that share the first j bits of their number.

Extensible hashing: two ideas

(a) Use *i* of *b* bits output by hash function



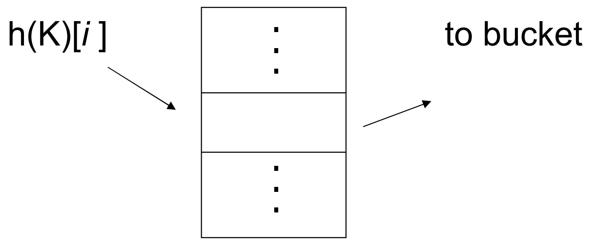
b

h(K) →

00110101

use *i* → grows over time….

(b) Use directory



h(K)[*i*]

to bucket

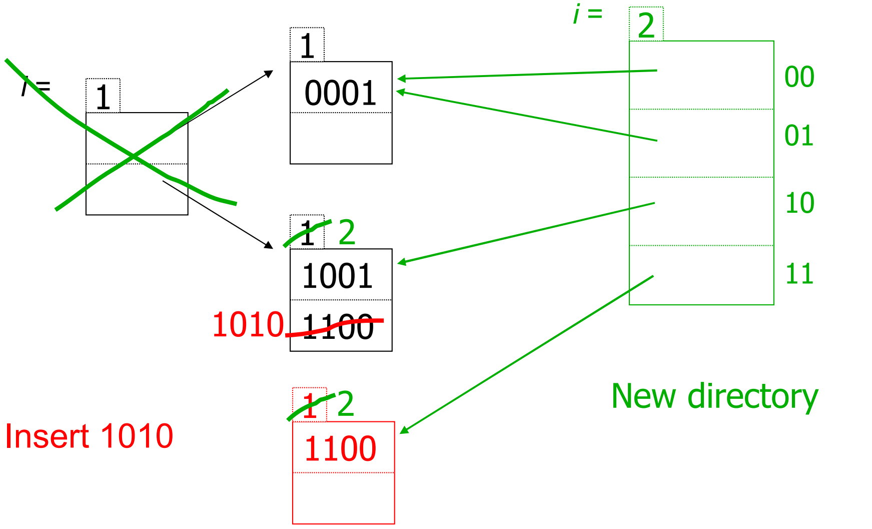
Extensible Hashtable Insert

If record with key K fits in the block pointed to by h(K), put it there.

If not, let this block represent j bits.

- Case 1: j < i: Split block according to (j + 1)st bit; set j := j + 1.
- Case 2: j = i: Set i := i + 1; split bucket array; proceed as in (1).

Example: h(k) is 4 bits; 2 records/block



~~*i* = 1~~

1
0001

~~1~~ 2
1001
1010 ~~1100~~

~~1~~ 2
1100

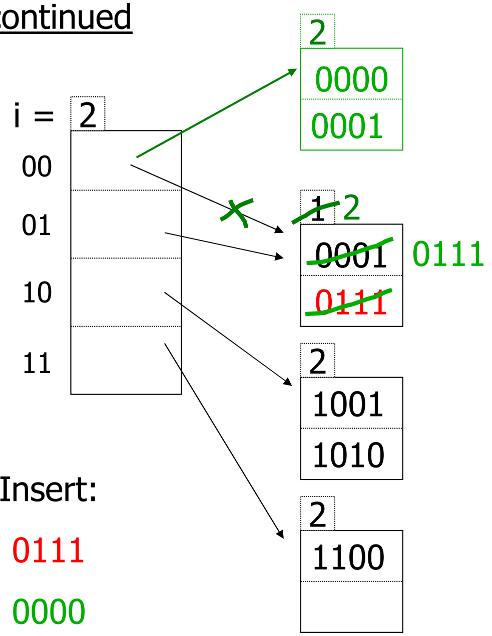
Insert 1010

i = 2

00
01
10
11

New directory

Example continued

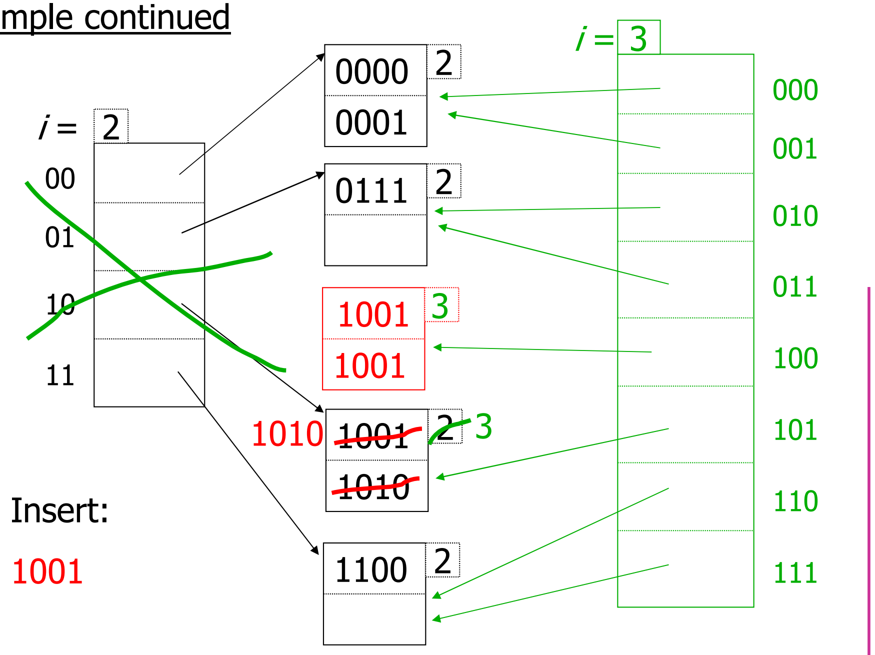

i = 2
00
01
10
11
2
0000
0001
~~1~~ 2
~~0001~~ 0111
~~0111~~
2
1001
1010
2
1100

Insert:

0111

0000

Example continued


i = 2
00
01
10
11
i = 3
000
001
010
011
100
101
110
111
0000
0001
2
0111
2
1001
1001
3
1010
~~1001~~
~~1010~~
~~2~~ 3
1100
2

Insert:

1001

Summary Extensible Hashing

+ Can handle growing files

- with less wasted space
- with no full reorganizations

- Indirection

(Not bad if directory in memory)

- Directory doubles in size