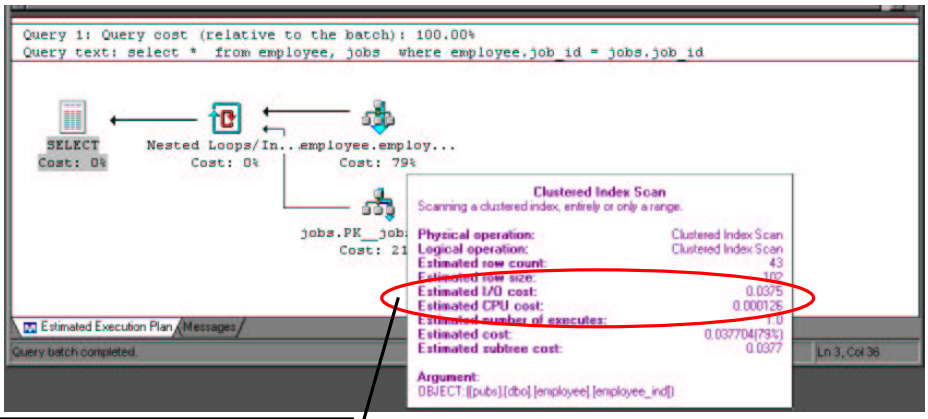
CSE 326: Data Structures
Lecture #14
More, Bigger Hash Please

Bart Niswonger
Summer Quarter 2001

Today's Outline

- Project
  - Rules of competition
  - Comments?
- Hashing
  - Probing continued
  - Rehashing
  - Extendible Hashing
  - Case Study

Cost of a Database Query



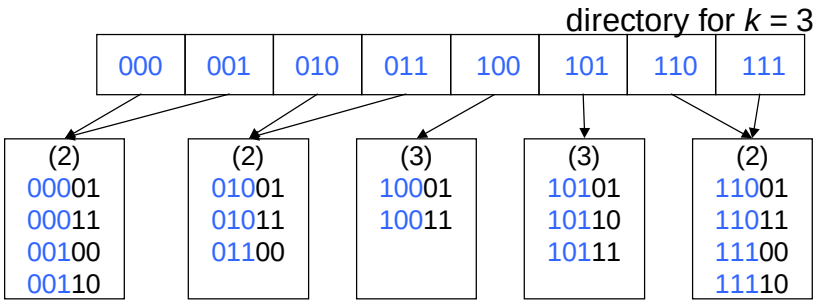
I/O to CPU ratio is 300!

Extendible Hashing

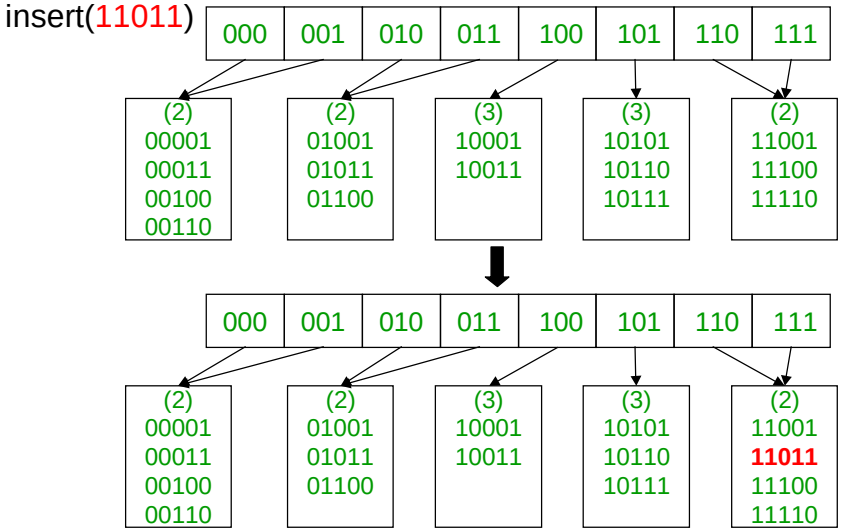
- Hashing technique for huge data sets
  - optimizes to reduce disk accesses
  - each hash bucket fits on one disk block
  - better than B-Trees if order is not important
- Table contains
  - buckets, each fitting in one disk block, with the data
  - a directory that fits in one disk block used to hash to the correct bucket

Extendible Hash Table

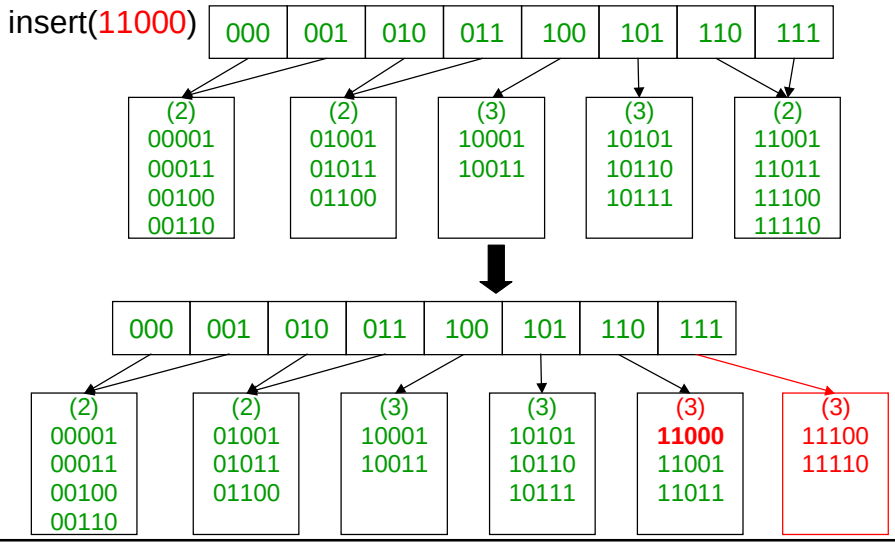
- Directory - entries labeled by *k* bits & pointer to bucket with all keys starting with its bits
- Each block contains keys & data matching on the first $j \leq k$ bits



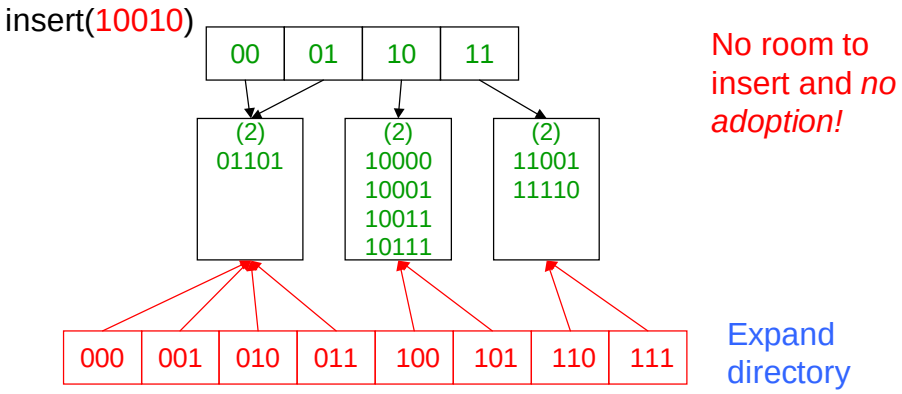
Inserting (easy case)



Splitting



Rehashing



Now, it's just a normal split.

When Directory Gets Too Large

- Store only pointers to the items
  - + (potentially) much smaller M
  - + fewer items in the directory
  - – one extra disk access!
- Rehash
  - + potentially better distribution over the buckets
  - + fewer unnecessary items in the directory
  - – can't solve the problem if there's simply too much data
- What if these don't work?
  - – use a B-Tree to store the directory!

Rehash of Hashing

- Hashing is a great data structure for storing unordered data that supports insert, delete & find
- Both separate chaining (open) and open addressing (closed) hashing are useful
  - – separate chaining flexible
  - – closed hashing uses less storage, but performs badly with load factors near 1
  - – extendible hashing for very large disk-based data
- Hashing pros and cons
  - + very fast
  - + simple to implement, supports insert, delete, find
  - - lazy deletion necessary in open addressing, can waste storage
  - - does not support operations dependent on order: min, max, range

Case Study

- Spelling dictionary
  - 30,000 words
  - static
  - arbitrary(ish) preprocessing time
- Goals
  - fast spell checking
  - minimal storage

- Practical notes
  - almost all searches are successful
  - words average about 8 characters in length
  - 30,000 words at 8 bytes/word is 1/4 MB
  - pointers are 4 bytes
  - there are *many* regularities in the structure of English words

Why?

Solutions

- Solutions
  - sorted array + binary search
  - open hashing
  - closed hashing + linear probing

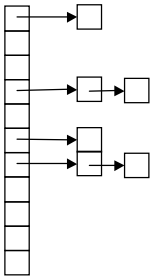
What kind of hash function should we use?

Storage

- Assume words are strings & entries are pointers to strings

Array + binary search

Open hashing



Closed hashing (open addressing)

How many pointers does each use?

Analysis

Binary search

- storage: n pointers + words = 360KB
- time: $\log_2 n \leq 15$ probes/access, worst case

Open hashing

- storage: $n + n/\lambda$ pointers + words ($\lambda = 1 \Rightarrow$ 600KB)
- time: $1 + \lambda/2$ probes/access, average ($\lambda = 1 \Rightarrow 1.5$)

Closed hashing

- storage: n/λ pointers + words ($\lambda = 0.5 \Rightarrow$ 480KB)
- time: $\frac{1}{2}\left(1+\frac{1}{(1-\lambda)}\right)$ probes/access, average ($\lambda = 0.5 \Rightarrow 1.5$)

Which one should we use?

To Do

- Start Project III (only 1 week!)
- Start reading chapter 8 in the book

Coming Up

- Disjoint-set union-find ADT
- Quiz (Thursday)