CSE 326: Data Structures
Topic #10: Hashing (3)

Ashish Sabharwal
Autumn, 2003

Today’s Outline

• Admin:
  – Handcny turnin for Project 2 – now!
  – Homework 2 due Friday
  – Start looking for a partner for Project 3
    (must be someone different from your Project 2 partner)

• Finish Hashing
  – Double hashing, rehashing
  – Extendible hashing

• Group Quiz #4

When to Rehash?

Many alternatives:
• Rehash when table is half full
• Rehash when insertion fails in open addressing
• Rehash when insertion becomes very slow in separate chaining
• Rehash when $\lambda$ crosses a certain threshold

Something We Again Forgot:
Disk Accesses

We Want To Minimize Disk Accesses!

Disk access time = Seek time + Transfer time

• Entire blocks transferred into memory at a time
• Transfer time much less than seek time
• Therefore we need to minimize disk accesses!

Solution: Extendible Hashing

Hashing technique for huge data sets
  – Optimizes to reduce disk accesses

Hash “table” contains
1. Directory
   2^p entries, $D$ bits per entry, pointers to leaf buckets
2. Leaf Buckets
   Keys in leaf $L$ have $d_L \leq D$ bits in common with parent key,
   leaves store all data

Properties
  – Only 2 levels in the table – only 2 disk accesses for find!
  – Each leaf bucket fits on one disk block – caching
  – Better than B-Trees if order is not important – why?
Extendible Hash Table

Directory entry:
key prefix (first \(D\) bits) and a pointer to the bucket
with all keys starting with that prefix

Bucket entry:
keys matching on first \(d_L\) \(\leq\) \(D\) bits, plus the data
associated with those keys

Bucket size = 4

Inserting Using Bucket-Split

Directory for \(D = 3\)

insert(1101)?
insert(1101)?

Inserting Using Directory-Expansion

1. insert(1001)
   But, no room to insert, only one parent, and no adoption!

2. Solution:
   Expand directory
   Now do a bucket-split

More expensive!

How to ensure this is uncommon?

What if Extendible Hashing Doesn’t Cut It?

Option 1: Store only pointers/references to the items:
(key, value) pairs separately on disk

Option 2: Improve hash function; Rehash

The One-Slide Hash

Collision resolution:
1. Separate Chaining
   – Expand beyond hashable via secondary Dictionaries
   – Allows \(\lambda > 1\)

2. Open Addressing
   – Expand within hashable
   – Secondary probing: linear, quadratic, double hash
   – \(\lambda \leq 1\) (by definition?)
   – \(\lambda \leq 1/2\) (by preference?)

Choosing a Hash Function
• Make sure table size is prime
• Careful choice for strings
• “Perfect hashing”
  – If keys known in advance; tune hash function for them

Rehashing
• Tunes up hashable when, e.g., \(\lambda\) causes a threshold

Extendible hashing
• For disk-based data

Search ADT Implementations

<table>
<thead>
<tr>
<th></th>
<th>insert</th>
<th>find</th>
<th>delete</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unsorted list</td>
<td>(\Theta(1))</td>
<td>(\Theta(n))</td>
<td>(\Theta(n))</td>
</tr>
<tr>
<td>Sorted list</td>
<td>(\Theta(n))</td>
<td>(\Theta(\log n))</td>
<td>(\Theta(n))</td>
</tr>
<tr>
<td>Trees</td>
<td>(\Theta(\log n))</td>
<td>(\Theta(\log n))</td>
<td>(\Theta(\log n))</td>
</tr>
<tr>
<td>Hash Table</td>
<td>(\Theta(1))</td>
<td>(\Theta(1))</td>
<td>(\Theta(1))</td>
</tr>
</tbody>
</table>

(average case)

Is there anything a hash table cannot do efficiently?

You’ll answer this in quiz #4!