CS222: Principles of Data Management

Notes #7
Static Hashing, Extendible Hashing

Instructor: Chen Li
## Cost of Operations

<table>
<thead>
<tr>
<th></th>
<th>(a) Scan</th>
<th>(b) Equality</th>
<th>(c) Range</th>
<th>(d) Insert</th>
<th>(e) Delete</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1) Heap</td>
<td>BD</td>
<td>0.5BD</td>
<td>BD</td>
<td>2D</td>
<td>Search +D</td>
</tr>
<tr>
<td>(2) Sorted</td>
<td>BD</td>
<td>Dlog 2B</td>
<td>D(log 2 B + # pgs with match recs)</td>
<td>Search + BD</td>
<td>Search +BD</td>
</tr>
<tr>
<td>(3) Clustered</td>
<td>1.5BD</td>
<td>Dlog F 1.5B</td>
<td>D(log F 1.5B + # pgs w. match recs)</td>
<td>Search + D</td>
<td>Search +D</td>
</tr>
<tr>
<td>(4) Unclust. Tree index</td>
<td>BD(R+0.15)</td>
<td>D(1 + log F 0.15B)</td>
<td>D(log F 0.15B + # pgs w. match recs)</td>
<td>Search + 2D</td>
<td>Search + 2D</td>
</tr>
</tbody>
</table>

Several assumptions underlie these (rough) estimates!
Introduction

- **Hash-based** indexes are best for *equality selections*. **Cannot** support range searches.
- Static and dynamic hashing techniques exist; trade-offs similar to ISAM vs. B+ trees.
Static Hashing

- # primary pages fixed, allocated sequentially, never de-allocated; overflow pages if needed.
- \( h(k) \mod M = \) bucket (page) to which data entry with key \( k \) belongs. \((M = \# \text{ of buckets})\)
Static Hashing (Contd.)

- Buckets contain *data entries*.
- Hash fn works on *search key* field of record $r$. Must distribute values over range $0 \ldots M-1$.
  - $h(key) = (a \ast key + b)$ usually works well.
  - $a$ and $b$ are constants; lots known about how to tune $h$.
- Long overflow chains can develop and degrade performance.
  - *Extendible and Linear Hashing*: Dynamic techniques to fix this problem.
Extendible Hashing

- Situation: Bucket (primary page) becomes full. Why not re-organize file by doubling # of buckets?
  - Reading and writing all pages is expensive!
  - **Idea**: Use directory of pointers to buckets, double # of buckets by doubling the directory, but splitting just the one data bucket that overflowed!
  - Directory much smaller than file, so doubling it is much cheaper. Only one page of data entries is split. *No overflow page now!*
  - Trick lies in how hash function is adjusted!
Example

- Directory is array of size 4.
- To find bucket for \( r \), take last `global depth' # bits of \( h(r) \); we denote \( r \) by \( h(r) \).
  - If \( h(r) = 5 = \text{binary } 101 \), it is in bucket pointed to by the two bits 01.
- **Insert**: If bucket is full, **split** it (allocate new page, re-distribute).
- **If necessary**, double the directory. (As we will see, splitting a bucket does not always require doubling; we can tell by comparing *global depth* with *local depth* for the split bucket.)
Insert $h(r) = 20$ (Causes Doubling)

- LOCAL DEPTH
- GLOBAL DEPTH
- DIRECTORY

Old Bucket A
4* 12* 32* 16*

Bucket A2 (`split image' of Bucket A)
4* 12* 20*

Bucket A
32* 16*

Bucket B
1* 5* 21* 13*

Bucket C
10*

Bucket D
15* 7* 19*

(left) (Needs to become 3 now)

(left) (Needs to become 3 now)
Points to Note

- 20 = binary 10100. Last 2 bits (00) tell us \( r \) belongs in A or A2. Last 3 bits needed to tell which one.
  - Global depth of directory: Max # of bits needed to tell which bucket an entry belongs to.
  - Local depth of a bucket: # of bits used to determine if an entry belongs to this bucket.

- When does bucket split cause directory doubling?
  - Before insert, local depth of bucket = global depth. Insert causes local depth to become > global depth; directory is doubled by copying it over and `fixing’ pointer to split page. (Note how use of least significant bits enables efficient doubling via bulk directory copying!)
Comments on Extendible Hashing

- If directory fits in memory, equality search answered with one disk access; else two.
  - 100MB file, 100 bytes/rec, 4K pages contains 1,000,000 records (as data entries) and 25,000 directory elements; chances are high that directory will fit in memory.
  - Directory grows in spurts, and, if the distribution of hash values is skewed, directory can grow large.
    - Multiple entries with same hash value cause problems!

- **Delete:** If removal of data entry makes bucket empty, can be merged with `split image`. If each directory element points to same bucket as its split image, can halve directory.