Indexes: Tree Based and Hash Based

B+-tree
Linear Hashing
B+ Tree: The Most Widely Used Index

- Insert/delete at ______ cost
  - keep tree *height-balanced*. \((F = \text{fanout}, \ N = \# \text{leaf}\ pages)\)

- Minimum 50% occupancy (except for root).
  - Each node contains \(d \leq m \leq 2d\) entries.
  - The parameter \(d\) is called the *order* of the tree.

- Supports equality and range-searches efficiently.

---

**Diagram:**

- **Index Entries** (Direct search)
- **Data Entries** ("Sequence set")
ICE: Inserting 35* …
ICE: Deleting 29*
ICE: Composite Search Keys

- B+-tree index on (Age, Salary)
- Which can you answer efficiently using a B+-tree?
  - Age = 20
  - Age > 20
  - Age = 20, Salary = 100000
  - Age > 20, Salary = 100000
  - Age = 20, Salary > 100000
  - Age > 20, Salary > 100000

- Assume B+-tree index on (Age, Salary, Bonus); which can you answer efficiently?
  - Age = 20, Salary = 100000, Bonus > 5000
  - Age = 20, Salary > 100000, Bonus > 5000
Extendible Hashing

• Main idea: If bucket (primary page) becomes full, why not re-organize file by *doubling* # of buckets?

  Essentially “splitting” buckets

• But reading and writing all buckets is expensive!
  – Idea: Use *directory of pointers to buckets*,
  – Double # of buckets by *doubling the directory*, splitting just the bucket that overflowed!
  – Directory much smaller than file, so doubling it is much cheaper.
  – *No overflow pages!*
ICE: Insert $h(r) = 20$
Comments on Extendible Hashing

• If directory fits in memory, equality search answered with ______ I/O; else ______
  – 100MB file, 100 bytes/rec, 4K pages contain 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 ________, directory can grow large
Linear Hashing

• This is another dynamic hashing scheme, an alternative to Extendible Hashing

• LH handles the problem of long overflow chains without using a directory, and handles duplicates

• Main idea:
Overview of LH File

- In the middle of a round.

Buckets that existed at the beginning of this round: this is the range of $h_{Level}$

Bucket to be split

Next

Buckets split in this round:
If $h_{Level}$ (search key value) is in this range, must use $h_{Level+1}$ (search key value) to decide if entry is in `split image' bucket.

`split image' buckets: created (through splitting of other buckets) in this round
ICE: Inserting $h(r) = 43$

Level=2, N=4

<table>
<thead>
<tr>
<th>h</th>
<th>h</th>
<th>PRIMARY PAGES</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>000</td>
<td>0</td>
<td>32<em>44</em>36*</td>
</tr>
<tr>
<td>001</td>
<td>1</td>
<td>9<em>25</em>5*</td>
</tr>
<tr>
<td>010</td>
<td>10</td>
<td>14<em>18</em>10<em>30</em></td>
</tr>
<tr>
<td>011</td>
<td>11</td>
<td>31<em>35</em>7<em>11</em></td>
</tr>
</tbody>
</table>

(This info is for illustration only!)
(The actual contents of the linear hashed file)

Next=0
Inserting $h(r) = 50$ (End of a Round)

<table>
<thead>
<tr>
<th>$h_3$</th>
<th>$h_2$</th>
<th>PRIMARY PAGES</th>
<th>OVERFLOW PAGES</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>00</td>
<td>32*</td>
<td></td>
</tr>
<tr>
<td>00</td>
<td>01</td>
<td>9* 25*</td>
<td></td>
</tr>
<tr>
<td>01</td>
<td>10</td>
<td>66* 18* 10* 34*</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>11</td>
<td>31* 35* 7* 11*</td>
<td>43*</td>
</tr>
<tr>
<td>10</td>
<td>00</td>
<td>44* 36*</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>01</td>
<td>5* 37* 29*</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>10</td>
<td>14* 30* 22*</td>
<td></td>
</tr>
</tbody>
</table>

Level=2

Next=3
Summary - Hashing

• Hash-based indexes: best for _______ searches, cannot support _____ searches.
• Static Hashing can lead to _________________.
• Extendible Hashing uses directory doubling to avoid _________________.
  – Duplicates may require _________________.
• Linear hashing avoids directory by splitting in rounds
  – Naturally handles _________________.
  – Uses overflow buckets (but not very long in practice)