More indexing

CPS 216
Advanced Database Systems

Outline

• Last time
  – The basics
  – ISAM
  – B+-trees and variants
• R-tree and variants
• Hash indexes
• Next time: inverted list, GiST

R-tree (SIGMOD 1984)

• B-tree: balanced hierarchy of 1-d ranges

• R-tree: balanced hierarchy of N-d regions
R-tree lookup

- Where am I?
  ![Diagram of R-tree lookup]

- Problem: search may go down many paths
  - Because regions may overlap
  - No performance guarantee like B-tree

R-tree insertion (slide 1)

Insert $R_9$ into R-tree

- Start from the root
- Pick a region containing $R_9$ and follow the child pointer
  - If none contains $R_9$, pick one and grow it to contain $R_9$
  - Pick the one that requires the least enlargement

R-tree insertion (slide 2)

- If a node is too full, split
  - Try to minimize the total area of bounding boxes
    - Quadratic: “seed” with the most wasteful pair; iteratively assign regions with strongest “preference”
    - Linear: “seed” with distant regions; iteratively assign others
R-tree insertion (slide 3)

- Split could propagate all the way up to the root (not shown in this example)

R*-tree (SIGMOD 1990)

- R-tree
  - Always tries to minimize the area of bounding boxes
  - Quadratic splitting algorithm encourages small seeds and possibly long and narrow bounding boxes
- R*-tree
  - Consider other criteria, e.g.
    - Minimize overlap between bounding boxes
    - Minimize the margin (perimeter length) of a bounding box
  - Forced reinserts
    - When a node overflows, reinsert “outer” entries
    - They may be picked up by other nodes, thus saving a split

R+-tree (VLDB 1987)

- Problem with R-tree
  - Regions may overlap
  - Search may go down many paths
- R+-tree
  - Regions in non-leaf nodes do not overlap
  - Search only goes down one path
  - But an insertion must now go down many paths!
    - R must be inserted into all R+-tree leaves whose bounding boxes overlap with R
Review

- Tree-structured indexes
  - ISAM
  - B-tree and variants
  - R-tree and variants
  - Can we generalize? GiST!
- Next: hash-based indexes

Static hashing

Performance of static hashing

- Depends on the quality of the hash function!
  - Best (hopefully average) case:
  - Worst case:
  - See Knuth vol. 3 for good hash functions
- Rule of thumb: keep utilization at 50%-80%
- How do we cope with growth?
  - Extensible hashing
  - Linear hashing
Extensible hashing (TODS 1979)

- Idea 1: use \( i \) bits of output by hash function and dynamically increase \( i \) as needed
  \[ h(k) \]
- Problem: \( ++i = \)
- Idea 2: use a directory
  - Just double the directory size
  - Many directory entries can point to the same bucket
  - Only split overflowed buckets
  “One more level of indirection solves everything!”

Extensible hashing example (slide 1)

- Insert \( k \) with \( h(k) = 0101 \)

Extensible hashing example (slide 2)

- Insert 1110, 0000

- Split again
  - No directory doubling this time
Extensible hashing example (slide 3)

- Insert 0001

Extensible hashing example (slide 4)

Delete? Just the reverse:
- -- local depth
merge buckets
- -- global depth if possible

Summary of extensible hashing

- Pros
  - Handles growing files
  - No full reorganization

- Cons
  - One more level of indirection
  - Directory size still doubles
  - Sometimes doubling is not enough!
Linear hashing (VLDB 1980)

- Grow only when utilization exceeds a threshold
- No extra indirection
  - Some extra math to figure out the right bucket

\[ i = 1 \quad \text{Number of bits in use} = \text{ceil}(\log_2 n) \]
\[ n = 2 \quad \text{Number of primary buckets} \]

Insert 0101
Threshold exceeded; grow!

| 0000 | 1111 |
| 1010 | 0101 |

20

Linear hashing example (slide 2)

- Grows linearly (hence the name)
- Split the \( (n - 2^{\lceil \log_2 n \rceil}) \)-th bucket (0-based index)
  - Intuitively, the first one with the lowest depth
  - Not necessarily the bucket being inserted into!

Insert 0001
Insert 1100
Threshold exceeded; grow!

| 0000 | 0001 | 1111 | 0101 | 1010 |
| 1100 |

| 1100 | 0101 | 1010 | 1111 |

\[ i = 2 \quad n = 3 \]
20

Linear hashing example (slide 3)

Insert 1110
Threshold exceeded; grow?

| 00 | 01 | 10 | 11 |
| 0000 | 0001 | 1010 | 1111 |
| 1100 | 0101 |

\[ i = 2 \quad n = 4 \]
21
Linear hashing example (slide 4)

- Look up 1110
  - 110 (6-th bucket) is not here
  - Then look in the \((6 - 2^{\text{floor} (\log_2 n)})\)-th bucket (= 2nd)

<table>
<thead>
<tr>
<th></th>
<th>0000</th>
<th>0010</th>
<th>1010</th>
<th>1110</th>
<th>1100</th>
</tr>
</thead>
</table>
| i = 3
| n = 5

Summary of Linear hashing

- Pros
  - Handles growing files
  - No full reorganization
  - No extra level of indirection
- Cons
  - Still has overflow chains
  - May not be able to split an overflow chain right away because buckets must be split in sequence

Hashing versus B-trees