Announcements

- Midterm grading completed
  - Problem 1,5: Yana
  - Problems 2,3,4 - Surajit
- HW#3 due today
- "Best 3" homeworks will be used for grades
  - It is in your benefit to turn in HW#4
- Reading list list for Last Wed 2/14 (Vol 2)
  - Section 3.1.3, 3.2, 3.3.1, 3.3.2, 3.4, 3.5, 4.1, 4.2

Announcements (2)

- Project Report Due next Wed
  - Project Web Page will be updated with guidelines
- Project Demo and Interview
  - March 6 (after class) and March 7
  - Yana will coordinate signing up
- Finals Overview – March 5
  - About 15 mins
  - Immediately after class
- Finals week office hours will be announced next week

B+ Trees

- Search trees
- Idea in B Trees:
  - make 1 node = 1 block
- Idea in B+ Trees:
  - Make leaves into a linked list (range queries are easier)

Indexing

Reading: Section 4.3, 4.4, 5.4 (Vol 2)

B+ Trees Basics

- Parameter d = the degree
- Each node has >= d and <= 2d keys (except root)

- Each leaf has >= d and <= 2d keys:
B+ Tree Example

\[ d = 2 \]

---

B+ Tree Design

- How large \( d \)?
- Example:
  - Key size = 4 bytes
  - Pointer size = 8 bytes
  - Block size = 4096 bytes
- \( 2d \times 4 + (2d+1) \times 8 \leq 4096 \)
- \( d = 170 \)

---

Searching a B+ Tree

- Exact key values:
  - Start at the root
  - Proceed down, to the leaf
- Range queries:
  - As above
  - Then sequential traversal

---

B+ Trees in Practice

- Typical order: 100. Typical fill-factor: 67%.
  - Average fanout = 133
- Typical capacities:
  - Height 4: \( 133^4 = 312,900,700 \) records
  - Height 3: \( 133^3 = 2,352,637 \) records
- Can often hold top levels in buffer pool:
  - Level 1 = 1 page = 8 Kbytes
  - Level 2 = 133 pages = 1 Mbyte
  - Level 3 = 17,689 pages = 133 MBytes

---

Insertion in a B+ Tree

Insert \((K, P)\)
- Find leaf where \( K \) belongs, insert
- If no overflow (2d keys or less), halt
- If overflow (2d+1 keys), split node, insert in parent:

---

Insertion in a B+ Tree

Insert \( K = 19 \)
Insertion in a B+ Tree

After insertion

Now insert 25

But now have to split!

After the split

Deletion from a B+ Tree

Delete 30
Deletion from a B+ Tree

After deleting 30
May change to 40, or not

Now delete 25

After deleting 25
Need to rebalance
Rotate

Now delete 40
Rotation not possible
Need to merge nodes

Final tree
Hash Tables

- Secondary storage hash tables are much like main memory ones
- Recall basics:
  - There are n buckets
  - A hash function \( f(k) \) maps a key \( k \) to \{0, 1, ..., n-1\}
  - Store in bucket \( f(k) \) a pointer to record with key \( k \)
- Secondary storage: bucket = block, use overflow blocks when needed

Hash Table Example

- Assume 1 bucket (block) stores 2 keys + pointers
- \( h(e) = 0 \)
- \( h(b) = h(f) = 1 \)
- \( h(g) = 2 \)
- \( h(a) = h(c) = 3 \)

Searching in a Hash Table

- Search for \( a \):
  - Compute \( h(a) = 3 \)
  - Read bucket 3
  - 1 disk access

Insertion in Hash Table

- Place in right bucket, if space
- E.g. \( h(d) = 2 \)

Insertion in Hash Table

- Create overflow block, if no space
- E.g. \( h(k) = 1 \)
- More overflow blocks may be needed

Hash Table Performance

- Excellent, if no overflow blocks
- Degrades considerably when number of keys exceeds the number of buckets (i.e. many overflow blocks).
**Extensible Hash Table**

- Allows hash table to grow, to avoid performance degradation
- Assume a hash function $h$ that returns numbers in $\{0, \ldots, 2^k - 1\}$
- Start with $n = 2^i \ll 2^k$, only look at first $I$ most significant bits

**E.g. $i=1$, $n=2$, $k=4$**

- Note: we only look at the first bit (0 or 1)

**Insertion in Extensible Hash Table**

- **Insert 1110**

- **Now insert 1010**
  - Need to extend table, split blocks
  - $i$ becomes 2

- **Now insert 1110**

- **Now insert 0000, then 0101**
  - Need to split block
Insertion in Extensible Hash Table

After splitting the block

Performance Extensible Hash Table

No overflow blocks: access always one read

BUT:
- Extensions can be costly and disruptive
- After an extension table may no longer fit in memory

Linear Hash Table

Idea: extend only one entry at a time
Problem: \( n \) no longer a power of 2
Let \( i \) be such that \( 2^i \leq n < 2^{i+1} \)
After computing \( h(k) \), use last \( i \) bits:
- If last \( i \) bits represent a number \( \geq n \), change msb from 1 to 0 (get a number \( \leq n \))

Linear Hash Table Example

\( N=3 \)

Linear Hash Tables

Extension: independent on overflow blocks
Extend \( n:=n+1 \) when average number of records per block exceeds (say) 80%
**Linear Hash Table Extension**

From $n=3$ to $n=4$

- Only need to touch one block (which one?)

From $n=3$ to $n=4$ finished

- Extension from $n=4$ to $n=5$ (new bit)
- Need to touch every single block (why?)
- Book is wrong here...

**Linear Hash Table Extension**

From $n=3$ to $n=4$ finished

- Extension from $n=4$ to $n=5$ (new bit)
- Need to touch every single block (why?)
- Book is wrong here...