CSE 250  
Lecture 37  
B+Trees
Fence Pointers Example

Binary Search: >273, ≤ 412

Array Index: 0 1 2 3 ...

keys 0 - 178 keys 192 - 273 keys 274-412 keys 458 - 611 ...

Page 0 Page 1 Page 2 Page 3

Load Page 2
ISAM Index

Binary Search @ Level 0
to find a Level 1 page

Binary Search @ Level 1
to find a Level 2 page

Binary Search @ Level 2
to find a Data page

Binary Search @ Data
to find the record

What does this look like?
ISAM Index

\[ n = C_{\text{data}} C_{\text{key}}^{\text{max} + 1} \]

\[ \frac{n}{C_{\text{data}}} = C_{\text{key}}^{\text{max} + 1} \]

\[ \log_{C_{\text{key}}} \left( \frac{n}{C_{\text{data}}} \right) = \text{max} + 1 \]

\[ \log_{C_{\text{key}}} (n) - \log_{C_{\text{key}}} (C_{\text{data}}) = \text{max} + 1 \]

Number of Levels: \[ O \left( \log_{C_{\text{key}}} (n) \right) \] = IO Complexity
What if the data changes?
Putting Data on Pages

- Page 0: keys 0 - 178
- Page 1: keys 192 - 273
- Page 2: keys 274 - 412
- Page 3: keys 458 - 611

Insert key 181
Putting Data on Pages

- **Idea:** Can keep “free” space on each page
  - ... what happens when the space fills up?
Putting Data on Pages

- Page 0: keys 0 - 178
- Page 1: keys 192 - 273
- Page 2: keys 274-412
- Page 3: keys 458 - 611

More keys 0 - 178
Putting Data on Pages

- **Idea:** Can keep “free” space on each page
  - ... what happens when the space fills up?
- **Idea:** Can use linked lists to store overflow
  - ... but that can makes the IO complexity O(n)
- **Idea:** Rearrange the tree
Dynamic Page Allocation

- Treat the disk as an ADT:
  - allocate(): PageID
    - Allocates a page in the data file and returns its position
  - load[T](page: PageID): T
    - Reads in a 4k chunk of data (e.g., T = Array[Byte])
  - write[T](page: PageID, data: T)
    - Writes a 4k chunk of data to the page
ISAM Index/Dynamic Page Allocation

Problem:
Keys at L0 no longer “line up” with pages at L1
Pointers to Pages
Pointers to Pages

“Pointers” (PageID)

Keys
Pointers to Pages

```
1 3 4 6
```
```
8 9 10 12
```
```
13 14 15 16
```
```
17 19 22 23
```
class DirectoryPage[K](size: Int)
{
    val entries = new Array[(PageID, K)](size)
}
Free Space Revisited

Add 9
Free Space Revisited

Add 14
Free Space Revisited

Add 10
Free Space Revisited

Add 11
Free Space Revisited

1 3 6

8 9 10 12

14 19 21

8 9 10 11 12
Free Space Revisited
Free Space Revisited
Free Space Revisited

Add 22, 27
Free Space Revisited

Diagram showing a tree structure with nodes labeled 6, 10, 12, and 21, and leaves labeled 1, 3, 6, 8, 9, 10, 11, 12, 14, 19, 21, 22, and 27.
Free Space Revisited
Free Space Revisited

```
1 3 6
8 9 10
11 12
```

```
12 27
```

```
6 10 12
```

```
21 27
```

```
14 19 21
```

```
22 27
```
Free Space Revisited

“B+ Tree” (Almost)
B+ Trees

• **Insert:**
  - Find the page that the record belongs on
  - Insert record there
  - If full, “split” the page
    • Insert additional separator in parent directory page
    • If full, “split” the directory page and repeat with parent
      - If “root split” create a new parent node
B+ Trees

- **Observation**: Don’t need the biggest key on each page
B+ Trees

... but what if $k > 12$?

... will never happen!
B+ Trees

Follow me if $k \leq 6$

Follow me if $6 < k \leq 10$

Follow me if $10 < k$
B+ Trees

- **Observation**: Don’t need the biggest key
- **Question**: What if the separator value is mispositioned?
B+ Trees

Add 17
B+ Trees

Diagram of a B+ tree with keys 1, 3, 6, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17.
B+ Trees
B+ Trees

- **Observation**: Don’t need the biggest key
- **Question**: What if the separator value is mispositioned?
  - **Idea**: “Steal” space from adjacent nodes
- **Question**: What happens when we delete records?
B+ Trees

Delete 27, 22
B+ Trees
B+ Trees

Delete 8-12
B+ Trees

$O(\log(n))$ reads required per search for the biggest $n$ in the tree’s history
B+ Trees

- **Observation**: Don’t need the biggest key
- **Question**: What if the separator value is mispositioned?
  - **Idea**: “Steal” space from adjacent nodes
- **Question**: What happens when we delete records?
  - **Observation**: The tree becomes unbalanced
    - **Idea**: “Minimum Fill”
B+ Trees Minimum Fill

• Each directory & data node must have at least $c/2$ records
  - Exception: The root
• 1 page read at level 1: $1/c/2$ of the list left to search
• 1 more page read at level 2: $1/c^2/4$ of the list left to search
• 1 more page read at level 3: $1/c^3/8$ of the list left to search
• Max tree depth:
  - $O(\log_{c/2}(N))$
B+ Trees

UNDERFULL

1 3 6

8 9 10

11 12

14 19 21
B+ Trees
B+ Trees

```
1 3 6
8 9 10
11 12
```

```
14 19 21
```
B+ Trees

• **Delete:**
  - Find the page that the record is on
  - Delete record (if present)
  - If underfull, “merge” the page with a neighbor
    • If either neighbor at > \( c/2 \) entries (can’t merge)
      - “steal” entries from neighbor
    • If parent underfull, “merge” parent with neighbor
      - Repeat as needed
      - If “root merge” drop lowest layer