**B-Trees**

CSE 326
Data Structures
Lecture 10

---

**Need for Multi-way Search**

- In very large databases nodes may reside on disk.
- The unit of disk access is a page, 1k, 2k or more bytes.

![Diagram](image)

---

**Example**

- 1k byte page
- Key 8 bytes, pointer 4 bytes
- \((M-1)8 + 4M = 1024\)
  \(12 M = 1032\)
  \(M = \lceil 1032/12 \rceil = 86\)

---

**B-Trees are multi-way search trees commonly used in database systems or other applications where data is stored externally on disks and keeping the tree shallow is important.**

A B-Tree of order \(M\) has the following properties:
1. The root is either a leaf or has between 2 and \(M\) children.
2. All nonleaf nodes (except the root) have between \(\lceil M/2 \rceil\) and \(M\) children.
3. All leaves are at the same depth.

---

**Example**

- B-tree of order 3 has 2 or 3 children per node

![Diagram](image)

---

**B-Tree Details**

Each (non-leaf) internal node of a B-tree has:
- Between \(\lceil M/2 \rceil\) and \(M\) children.
- up to \(M-1\) keys \(k_1 < k_2 < \ldots < k_{M-1}\)

![Diagram](image)

**Keys are ordered so that:**

\(k_1 < k_2 < \ldots < k_{M-1}\)
B-Tree Details

Each leaf node of a B-tree has:
- Between \( \lceil M/2 \rceil \) and \( M \) keys and pointers.

Keys are ordered so that:
\[ k_1 < k_2 < \cdots < k_M. \]

Keys point to data on other pages.

Properties of B-Trees

Children of each internal node are “between” the items in that node. Suppose subtree \( T_i \) is the \( i \)-th child of the node:
- all keys in \( T_i \) must be between keys \( k_{i-1} \) and \( k_i \).
- \( k_{i-1} \) is the smallest key in \( T_i \).
- All keys in first subtree \( T_1 < k_1 \).
- All keys in last subtree \( T_M \geq k_M. \)

Example: Searching in B-trees

- B-tree of order 3: also known as 2-3 tree (2 to 3 children)

```
       13:-
      6:11
  3 4 6 7 8 11 12 13 14 17 18
```

- Examples: Search for 9, 14, 12
- Note: If leaf nodes are connected as a Linked List, B-tree is called a B+ tree — Allows sorted list to be accessed easily

Inserting into B-Trees

- Insert X: Do a Find on X and find appropriate leaf node
  - If leaf node is not full, fill in empty slot with X
  - E.g. Insert 5
  - If leaf node is full, split leaf node and adjust parents up to root node
    - E.g. Insert 9

```
       13:-
      6:11
  3 4 6 7 8 11 12 13 14 17 18
       8 9
```

```
       13:-
      6:11
  3 4 6 7 8 9 11 12 13 14 17 18
```

```
       13:-
      6:11
  3 4 6 7 8 9 11 12 13 14 17 18
```

```
       13:-
      6:11
  3 4 6 7 8 9 11 12 13 14 17 18
```

```
       13:-
      6:11
  3 4 6 7 8 9 11 12 13 14 17 18
```

```
       13:-
      6:11
  3 4 6 7 8 9 11 12 13 14 17 18
```

```
       13:-
      6:11
  3 4 6 7 8 9 11 12 13 14 17 18
```

```
       13:-
      6:11
  3 4 6 7 8 9 11 12 13 14 17 18
```

```
       13:-
      6:11
  3 4 6 7 8 9 11 12 13 14 17 18
```

```
       13:-
      6:11
  3 4 6 7 8 9 11 12 13 14 17 18
```

```
       13:-
      6:11
  3 4 6 7 8 9 11 12 13 14 17 18
```

```
       13:-
      6:11
  3 4 6 7 8 9 11 12 13 14 17 18
```

```
       13:-
      6:11
  3 4 6 7 8 9 11 12 13 14 17 18
```

```
       13:-
      6:11
  3 4 6 7 8 9 11 12 13 14 17 18
```

```
       13:-
      6:11
  3 4 6 7 8 9 11 12 13 14 17 18
```

```
       13:-
      6:11
  3 4 6 7 8 9 11 12 13 14 17 18
```

```
       13:-
      6:11
  3 4 6 7 8 9 11 12 13 14 17 18
```

```
       13:-
      6:11
  3 4 6 7 8 9 11 12 13 14 17 18
```

```
       13:-
      6:11
  3 4 6 7 8 9 11 12 13 14 17 18
```
Insert Example

Deleting From B-Trees

- Delete X: Do a find and remove from leaf
  - Leaf underflows — borrow from a neighbor
    - E.g. 11
  - Leaf underflows and can’t borrow — merge nodes, delete parent
    - E.g. 17

Delete Example

Delete Example

Delete Example

Run Time Analysis of B-Tree Operations

- For a B-Tree of order M
  - Each internal node has up to M-1 keys to search
  - Each internal node has between $\lceil M/2 \rceil$ and M children
  - Depth of B-Tree storing N items is $O(\log \lceil M/2 \rceil N)$
- Example: $M = 86$
  - $\log_{43} N = \log_{43} N / \log_{43} 43 = 1.84 \log_{43} N$
  - $\log_{43} 1,000,000,000 = 5.51$
Summary of Search Trees

- Problem with Search Trees: Must keep tree balanced to allow fast access to stored items
- AVL trees: Insert/Delete operations keep tree balanced
- Splay trees: Repeated Find operations produce balanced trees on average
- Multi-way search trees (e.g. B-Trees): More than two children per node allows shallow trees; all leaves are at the same depth keeping tree balanced at all times