Motives

- Large-scale databases are stored in disks/hard drives.
- Disks are quite different from main memory.
  - Data in a disk are accessed through a read-write head.
  - To read a piece of data, the read-write head must first be positioned right before the data and then scans through the data.
  - Moving the read-write head is terribly slow, compared to the speed of the processor.
  - Implication:
- What would happen if we implement AVL trees on disks?

A Wish List

To store a large amount of data in disks, we need is a tree structure that has the following properties.

1. Arrange data in a way similar to BSTs to enable efficient searching.
2. Balanced, not necessarily in the sense of the AVL trees.
3. Work well with large tree nodes.
Definition

Given a positive constant integer, called MINIMUM, a B–tree satisfies the following rules.

1. The root may have as few as zero/one entry; every other node has at least MINIMUM entries.

2. The maximum number of entries in a node, denoted as MAXIMUM, is twice the value of MINIMUM.

3. The entries of each B–tree node are stored in an array in ascending order, with the smallest at index 0.

4. The number of subtrees below a non-leaf node is always one more than the number of entries in the node.

5. For any non-leaf node:
   (a) an entry at index \( i \) is greater than all the entries in subtree number \( i \) of the node.
   (b) an entry at index \( i \) is less than all the entries in subtree number \( i + 1 \) of the node.

6. Every leaf in a B–tree has the same depth.

Note: C++ implementations of B-trees will NOT be covered in this course.
An Example, MINIMUM = 2

Contents of a B–tree Node

- data count – number of items stored in this node
- subtree count – number of items stored in this node
  - at leaves, subtree count = 0
  - at other nodes, subtree count = data count + 1
- data[MINIMUM*2] – holds the data entries
- subtree[MINIMUM*2 + 1] – points to the data count + 1 subtrees
### Printing a B–Tree

if (subtree_count == 0)
    print out data[0] to data[data_count-1]
else {
    for i from 0 to data_count-1
    {
        print subtree[i] (recursively)
        print data[i]
    }
    print subtree[data_count] (recursively)
}

### Searching a B–tree

if (target = some data[k], 0 \leq k < data_count) return TRUE
else if (subtree_count == 0) return FALSE
else {
    k = (smallest i, if target < data[i]) OR
        (subtree_count-1, if target > the last entries)
    search in subtree[k]
}
Insertion to B-Trees

<table>
<thead>
<tr>
<th>15</th>
<th>25</th>
<th>32</th>
<th>20</th>
<th>40</th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td>25</td>
<td>32</td>
<td>25</td>
<td>40</td>
</tr>
<tr>
<td>45</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

MINIMUM = 3

- Only the root can sometimes hold less than MINIMUM entries; other nodes are created with at least MINIMUM entries.
- The two leaves are of the same depth right at their creation.

Splitting A Node

MAXIMUM entries

Insert

MAXIMUM + 1 entries

MINIMUM

MINIMUM

Split and promote the middle entry
Observations

- It is the middle entry that is promoted, not the newly inserted entry (they could be the same entry though).
- The promoted entry becomes the new root if the split node is the root of the entire B-tree.
- Otherwise, the promoted entry joins the parent node of the split node.

When A Promoted Entry Joins Its Parent

- After the promotion, the parent may have more than MAXIMUM entries and need to be split itself.
- This split-and-promote process could propagate all the way up to the root.
Let us insert 55 to

```
30
20  40
10, 15  25  35, 38  45, 50
```

After the Insertion of 55

```
30
20  40, 50
10, 15  25  35, 38  45  55
```

Next, let us insert 37.
Now, let us insert 5.
After the Insertion of 5

Next, insert 18.

After the Insertion of 18

Let us insert 12.
Promote 15

30, 40

10, 15, 20

15, 30, 40

10

5, 12, 18, 25, 35

15, 30, 40

5, 12, 18, 25, 35

15, 30, 40

5, 12, 18, 25, 35
Note that depths of leaves are identical all the time. The only time a B-tree grows is when its root is split.

Deletion from B-Trees

- Try not to change the height of the tree for as long as possible.
- When unavoidable, the heights of all nodes are reduced simultaneously.

The easy case: deleting 58 from the B-tree below (MINIMUM = 1)
After the Deletion of 58

A more difficult case: deleting 65.

Borrow from the Nearest Right Sibling

Condition: the nearest right sibling must have at least \( \text{MINIMUM}+1 \) entries.

Can you figure out how to borrow from the nearest left sibling?
After the Deletion of 65

Next, delete 55.

Merge with the Nearest Right Sibling

**Condition:** The nearest right sibling has exactly MINIMUM entries.

Can you figure out how to merge with the nearest left sibling?
Observations

- If the merge leaves the parent with only MINIMUM-1 entries, than we need to fix the shortage problem of the parent too (by borrowing or merging).
- This merge process could propagate all the way up to the root.

After the Deletion of 55

Lastly, let us delete 40.
Deletion from a Non-Leaf Node

Swap the deletion target with the maximum entry in the “left” subtree of the target.

Example: to delete

```
50
/     \
20,30 60,70
/       \
5,10 11,12 48,49 55,58 65 75,80
```
Performance Analysis

Consider a B-tree with MINIMUM=10.

- What is the minimum number of entries in the tree when the height of the tree is 2?

- Repeat the above problem for heights 3, 4, 5, and $h$.

- What is the maximum height for B-trees that contain 1 billion entries?

Counting Disk Accesses

- Let $h$ be the height of a given B-tree.
- In the worst case, an insertion can require $2h$ disk accesses.
  - exactly $h$ accesses in the downward, searching process
  - at most $h$ accesses in the upward, splitting process
- In the worst case, a deletion can also require $2h$ disk accesses.
  - exactly $h$ accesses in the downward, searching process
  - at most $h$ accesses in the upward, merging process
- It is possible to reduce the worst cases to $h$ for both insertion and deletion, by allowing MAXIMUM=$2$*MINIMUM+1.
  - such B-tree variations are outside the scope of this course