Insertions and deletions in B+ trees

Introduction to Database Design 2011, Lecture 11
Supplement to lecture slides
Initial setup

• We consider the B+ tree below
Observations

• Observe that the tree has fan out 3

• Invariants to be preserved
  - Leafs must contain between 1 and 2 values
  - Internal nodes must contain between 2 and 3 pointers
  - Root must have between 2 and 3 pointers
  - Tree must be balanced, i.e., all paths from root to a leaf must be of same length
Inserting Vince

Rasmus Ejlers Møgelberg
Inserting Vera

- Leaf consisting of tom and vince is split and extra pointer is inserted in parent
Inserting rob

- Inserting rob is more difficult. We first create a new leaf node and insert it as below.

- The node above is temporarily extended to contain 4 pointers.

```
    judy   rick
  /       \
bob   jane
    / \
   /   \
  /     \
abo el   bob eddie
```

```
    mike  pete
  /       \
sol   tom
    /       \
vince
  /
   /
  /  
sol rob
    / \
  /   \
  /     \
abo el     tom vera
```

```
    karen  nan
  /       \
pete   phil
    /       \
rick rob
  /       \
  /       \
abo el     tom vera
```

```
    al
abo el
  / \
  /   \
  /     \
abo el
```

```
    al
abo el
  / \
  /   \
  /     \
abo el
```
Inserting rob

- The overfull internal node is then split in 2
- The new pointer is inserted into the root node which then becomes overfull
Inserting rob

- Finally the overfull root is split in 2
- At this point the tree satisfies the requirements of slide 3 and so the insertion procedure ends
Deleting jane

- Is straight forward
- Note that the node above the leaf where jane was deleted must also be updated
Deleting bob
Deleting joe

- Leads to a leaf being deleted and the parent being updated
Deleting eddie

- Leads to deletion of a leaf
- At this point the parent becomes underfull
Deleting eddie

- When a node becomes underful the algorithm will try to **redistribute** some pointers from a neighbouring sibling to it.

- Since this is possible in this case we do it.
Deleting abe and al

- Leads to deletion of a leaf
- This makes the parent underfull.
- We cannot redistribute pointers again since this will make the neighbour underfull.
Deleting abe and al

• Instead we must **merge** with the neighbouring sibling

• But this makes the parent underfull
Deleting abe and al

- Since we can not solve this problem by redistributing pointers we must merge siblings again.
Deleting abe and al

- Since the root is underfull it can be deleted
- The resulting tree satisfies the requirements and so the deletion algorithm ends
General remarks

- When a node becomes underfull the algorithm will try to redistribute pointers from the neighbouring sibling either on the left or the right

- If this is not possible, it should merge with one of them

- The value held in an internal node or the root should always be the smallest value appearing in a leaf of the subtree pointed to by the pointer after the value