Deletion from a B-tree

(We assume search keys in B-tree are unique).

Delete Algorithm (delete x)

1. Find the leaf node L where search key x belongs
   - If not found: done
   - If found: delete (x, pointer(x))

2. If after deletion:
   - L has $\geq \left\lceil \frac{n+1}{2} \right\rceil$ (min) keys

   $\Rightarrow$ Done

3. If L underflow:

   $\Rightarrow$ transfer

   $\Rightarrow$ merge
Underflow: a node contains less than the min. # keys/pointers

Methods to solve underflow:

1) preferred (1st choice) solution: transfer

If either sibling has > min. # keys then:

- transfer one (key, pointer) to the underflow node.

* Parent node need to adjust a key search value (transferred key is stored in parent)
(2) Method at last resort: Merge with one sibling.

- Half full
- Underflow node

Left Merge

Sibling nodes [both sibling nodes have min. # keys]

Right merge

Merged node will have max # keys.

Full node.

The parent node will lose one (key, pointer) pair (because it has one less child node!!!)
Propagated Merge/Delete

Note: Because the parent node will lose one key, pointer

\[ \text{Parent node can undergo underflow in a merge operation.} \]

\[ \text{(Parent node undergoes a deletion)} \]

\[ \text{Handle underflow in parent with} \]
\[ \text{transfer} \]
\[ \text{or: another merge} \]
\[ \text{Can propagate all the way to the root!!} \]
Example: delete - simple case

Delete in a node that does not result in underflow.

E.g.:

13 17 29

delete (17)

\[ \text{shift \ keys + pointers!} \]

\[ 13 \ 29 \ 29 \]

\( \times \) You must use the \textbf{SEARCH alg.}

to find the key in the list

\textbf{FIRST} (prep step)
delete(17)

Result:

Need to move keys and pointers.

13 29
A  C
Delete: medium-hard case: transfer.

Delete (7)

2 update parent search key !!!

\[ \begin{array}{c}
\text{transfer}
\end{array} \]

\[ n = 3, \quad \left\lceil \frac{n+1}{2} \right\rceil = \left\lceil \frac{4}{2} \right\rceil = 2 \]

1 key left, \( \geq 2 \) keys mm

\( \Rightarrow \) Underflow

Sibling has 3 keys \( > 2 \) keys

\( \Rightarrow \) transfer.
**Delete(7)**

Diagram showing a B-tree with keys 2, 3, 5, 7, 11, 13, 17, 29, 31, 37, 41, 43, 47. The process of deleting a node with key 7 is illustrated.

1. **After delete:**
   - B 1 key < 2 (mm).
   - Underflow.

2. **Transfer from sibling:**
   - Parent must be updated!
After transfer, update key in parent node.

Done
Delete: hard case: merge

Start with tree after delete(7) (from last example)

Now: delete(11).

Diff. parents!!!

```
5
\[ 2, 3 \]
\[ 15, 14 \]
\[ 13, 17, 21 \]
```

```
P1
\[ 5 \]
\[ 2, 3 \]
\[ 15, 14 \]
\[ 13, 17, 21 \]
```

```
P2
```

\[ \text{NOT a sibling!} \]
\[ \text{(diff. parent!!)} \]

Underflow:

```
2, 3, 1, 1
15
```

\[ \text{min # of keys} \]
\[ \text{cannot transfer} \]

\[ \Rightarrow \text{Merge!!} \]
B-tree07b.gif (GIF Image, 858 x 457 pixels)

http://192.168.1.9/~cheung/teaching/web/577/Syllabus/GIFS/B-tree07...

(1) Before merge:

(2) After merge:

Deleted !!!

# phs < 2

Underflow

★ Copy this node !!!
Situation Now:

Underflow node

Solve with transfer

Note: transfer between internal node is different:

It is a 3-way transfer:

(Keys + pointers go to different nodes!!!)

Thrs is the "(a,b)-tree" transfer alg!
This figure shows where the
- keys are transferred
- where a pointer is transferred.
Result

```plaintext
X
Y
Z
```

![B-tree diagram](http://192.168.1.9/~cheung/teaching/web/577/Syllabus/GIFS/B-tree07e.gif)