The B-Tree

Yufei Tao

ITEE
University of Queensland

Before ascending into d-dimensional space $\mathbb{R}^d$ with $d > 1$, this lecture will focus on one-dimensional space, i.e., $d = 1$. We will review the B-tree, which is a fundamental structure that can be used to process many types of queries on one-dimensional points.

Range Reporting

Let S be a set of points in $\mathbb{R}$. Given an interval $q = [x, y]$, a range query returns $S \cap q$, namely, all the points in S that are covered by q.

Example

- Assume $S = \{1, 13, 17, 25, 36, 49, 52, 67\}$.
- For $q = [5, 20]$, the result is $\{13, 17\}$.
- For $q = [27, 30]$, the result is $\emptyset$.

Computation model

- We assume that the input set S does not fit in memory, and thus needs to be stored in the disk.
- The disk has been formatted into *blocks* (also called *pages*), such that each block has the same size, e.g., 4096 bytes.
- An *I/O* either reads a block from the disk into memory, or writes a block of memory to the disk.
- We measure the cost of an operation in the number of I/Os that need to be performed.

Naively, we can solve any range query by scanning the whole S, but the cost is obviously prohibitive. The B-tree allows us to do much better.

B-tree

- Each leaf node has between $B/2$ and B data elements, where B is a parameter that is at least 3. The only exception takes place when the leaf is the root, in which case it can have any number of elements. All the leaf nodes are at the same level.
- Each internal node has between $B/2$ and B child nodes, except that the root can have as few as 2 child nodes.

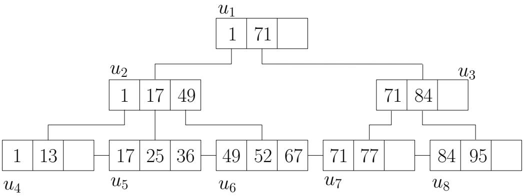
B-tree (cont.)

- For any node u, denote by S_u the set of elements in the subtree of u. Now let u be an internal node with child nodes $v_1, ..., v_f$ $(f \leq B)$.
  - All the data elements in S_{v_i} must be strictly smaller than any data element in S_{v_j} for any $1 \leq i < j \leq f$.
  - For each v_i $(i \leq f)$, u stores the smallest element $r(v_i)$ in S_{v_i}, which is referred to as a routing element.

Note that $r(u) = r(v_1)$.

Example

$B = 3$



In practice, the leaf nodes are connected in a doubly-linked list, sorted by their natural ordering.

Think: How to construct a B-tree if the dataset S has been sorted?

Successor and Predecessor Search

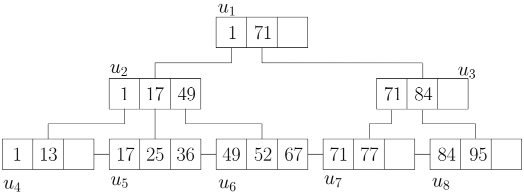
- Let S be a set of real values. The successor (or predecessor) of a value x in S is the smallest (or largest, resp.) value in S that is at least (most, resp.) x.

  For example, given $S = \{1, 13, 17, 25\}$, the successors of 10 and 13 are both 13, while the predecessor of 12 is 1.

- Given a B-tree T on S, we can find the successor of any x by visiting a single root-to-leaf path in T:
  1. Set u to the root of T.
  2. If u is a leaf, return the successor of x among the elements in u.
  3. Otherwise, find the predecessor of x among the (routing) elements in u. Let it be the $r(v)$ of some child node v of u. Set u to v and go to Step 2.

Example

u_1, u_2, u_5 are accessed to retrieve the successor of 20.



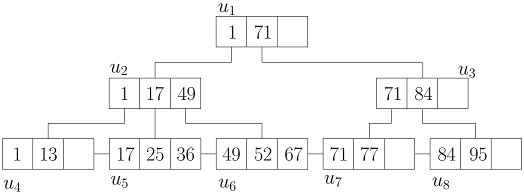
Cost = 3 I/Os.

Answering A Range Query $q = [x, y]$

1. Locate the leaf u containing the successor of x.
2. Report all elements in u that fall in q.
3. If no element in u is greater than y, set u to the succeeding leaf node, and go to Step 2.

Example

The algorithm accesses u_1, u_2, u_5, u_6, and u_7 to answer a range query with $q = [20, 75]$.



Think: How should the algorithm be adapted if leaf nodes are not linked together?

Insertion

To insert a value x into a B-tree:

1. Find the leaf node u that should accommodate x without violating the B-tree definition. Add x to u.
2. If u has no more than B elements, the insertion is complete. Otherwise, u verflows.

Overflow Handling

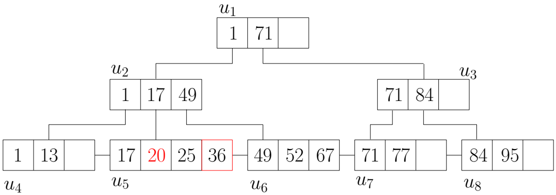
Let u be the node that overflows.

1. Create a new node u'.
2. Split the elements of u in two halves, respecting their ordering. Put the first (second) half in u (u').
3. Insert $r(u')$ into the parent p of u.
4. If p has no more than B elements, done. Otherwise, handle the overflow at p in the same way.

The changes may propagate all the way up. If the root splits, create a new root.

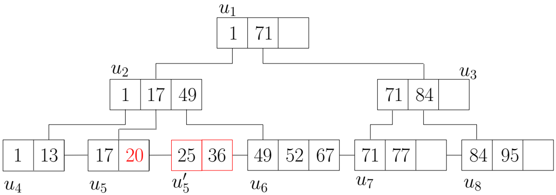
Example

The insertion of 20 makes u_5 overflow.



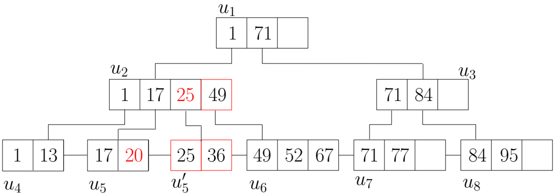
Example (cont.)

u_5 splits, and generates u_5'.



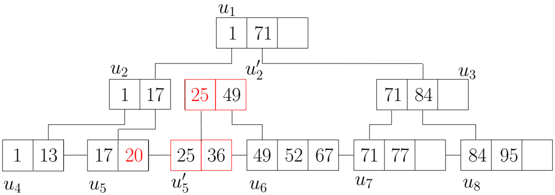
Example (cont.)

$r(u_5') = 25$ is added to u_2, which overflows.



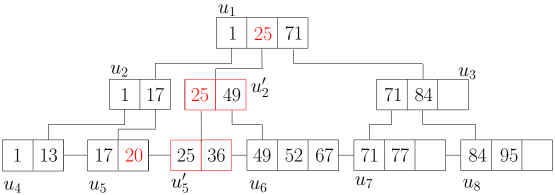
Example (cont.)

u_2 splits, and generates u_2'.



Example (cont.)

$p(u_2')$ is added the root. Done!



Deletion

To delete a value x from a B-tree:

1. Find the leaf node u that contains x. Remove x from u.
2. If x is the smallest element in u, adjust the routing elements in the ancestors of u appropriately.
3. If u is the root or has at least $B/2$ elements, the deletion is complete. Otherwise, u underflows.

Underflow Handling

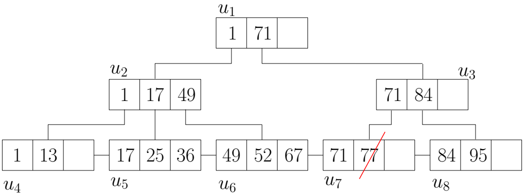
Let u be the node that underflows, and u' be the right sibling of u (if u does not have a right sibling, set u' to its left sibling, and swap u and u' in the below):

1. If u and u' contain no more than B elements in total, perform a merge:
   - Put all the elements in u.
   - Remove $r(u')$ from the parent p of u.
   - If p underflows, handle it in the same way.
2. Otherwise, perform a share:
   - Distribute the elements in u and u' equally between them.
   - Modify $r(u')$ in p.

If the root ends up having only one child, remove the root.

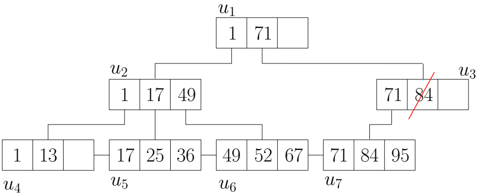
Example

Deletion of 77 makes u_7 underflow.



Example (cont.)

Merge u_7 and u_7', and remove $r(u_7') = 84$ from p_3, causing the underflow of u_3.



Example (cont.)

Perform a share between u_2 and u_3. Update $r(u_3) = 49$ in u_1.