Module 11: External Memory - enriched

T. Biedl  M. Petrick  O. Veksler
Based on lecture notes by many previous cs240 instructors

David R. Cheriton School of Computer Science, University of Waterloo

Winter 2019

References: Goodrich & Tamassia 20.1-20.3, Sedgewick 16.4
Outline

1. External Memory
   - Towards $B$-trees
   - Red-black trees
   - $B^+$-trees
   - Cache-oblivious trees
Outline

1 External Memory
   • Towards $B$-trees
   • Red-black trees
   • $B^+$-trees
   • Cache-oblivious trees
Towards $B$-trees

Balanced BST: Search path has length $\Theta(\log n)$
Might need $\Theta(\log n)$ block-transfers.

Idea: Store whole subtrees (say $b - 1$ KVPs) in one block.

$$\#\text{block-transfers} = \frac{\Theta(\log n)}{\log b} = \Theta(\log_b n)$$

But: how to insert efficiently?
Outline

1. External Memory
   - Towards $B$-trees
   - Red-black trees
   - $B^+$-trees
   - Cache-oblivious trees
Converting a 2-4-tree:

- A $d$-node becomes a black node with $d - 1$ red children
  (Assemble in the only possible way to have a BST.)

Resulting properties:

- Subtrees of red nodes are empty or have black root
- Any empty subtree $T$ has the same black-depth
  (number of black nodes on path from root to $T$)
Red-black tree

Red-black tree properties:
- Every node is red or black.
- Subtrees of red nodes are empty or have black root.
- Any empty subtree \( T \) has the same black-depth.

Converting to a 2-4-tree:
- Black node and \( 0 \leq d \leq 2 \) red children become a \((d + 1)\)-node.
Outline

1. External Memory
   - Towards $B$-trees
   - Red-black trees
   - $B^+$-trees
   - Cache-oblivious trees
Towards $B^+$-trees

There are two variants of tree-structures:

**Storage-variant:** Every node stores a KVP.

**Heap:**

```
    50,value
   /   \
29,value 34,value
  /   \   /   \
27,value 15,value 8,value 10,value
 /   \ /   \ /   \ /   \
23,value 26,value 23,value 29,value 50,value
```

**BST-tree:**

```
    15,value
   /   \
 6,value 25,value
 /   \   /   \
10,value 23,value 27,value 50,value
```

**Decision-variant:** All KVPs at leaves, internal nodes/edges guide search.

**Trie:**

```
  00$,value
  /   \
0 $ 1 $ 0001$,value
 /   \ /   \ /   \ /   \
$ 0 $ 11$,value $ 01001$,value
 /   \ /   \ /   \ /   \
0 1 $ 0 $ 01$,value 010$,value
 /   \ /   \ /   \ /   \
1 1 $ 1 $ 11$,value 110$,value
 /   \ /   \ /   \ /   \
$ 1 $ 0 $ 11$,value 110$,value
```

**kd-tree:**

```
    x < p_3 \cdot x ?
       /   \         /
   y < p_1 \cdot y ?  y < p_6 \cdot y ?
   /   \  /   \
 x < p_2 \cdot x ?  x < p_9 \cdot x ?  p_7 \cdot value  x < p_4 \cdot x ?
 /   \ /   \ /   \
p_0 \cdot value p_2 \cdot value p_3 \cdot value y < p_9 \cdot y ? y < p_8 \cdot y ?
 /   \ /   \ /   \ /   \
p_1 \cdot value p_9 \cdot value p_6 \cdot value p_8 \cdot value p_5 \cdot value p_4 \cdot value
```

Biedl, Petrick, Veksler (SCS, UW)
Towards $B^+$-trees

- For storage-variant, there usually exists an equivalent decision-variant.

For example for heaps:
- Choose tree with heap-structure and $n$ leaves, store items at leaves.
- Edges store maximum key in the subtree
- Routines need minor adjustment: no exchanges of KVPs

But in *internal memory*, this wastes space
(typically $\approx$ twice as many nodes)
$B^+$-trees

- $B^+$-tree: The decision-variant of a $B$-tree.
- $B$-tree of order 5:

  - $B^+$-tree of order 5:

  ▶ All KVPs are in the leaves
  ▶ Key at internal node holds minimum key from subtree to the right
  ▶ Search/insert need some minor adjustments (no details)

- We may not even need more nodes!
**Why do we do this?**
- Recall: keys (usually) fairly small, while values potentially *huge*
  - E.g. key = facebook user ID, value = all your posts + metadata
- Recall: Order of the $B$-tree is chosen such that a node fits into a block
  ⇒ In a $B^+$-tree, internal nodes can use *many* more keys than KVPs!
  ⇒ The height will be much smaller ⇒ fewer block transfers
Outline

1 External Memory
   • Towards $B$-trees
   • Red-black trees
   • $B^+$-trees
   • Cache-oblivious trees
Cache-oblivious trees

\[ T_0 \]

\[ \approx \sqrt{n} \text{ children} \]

\[ \approx \sqrt{n+1} \text{ trees with } \approx \sqrt{n} \text{ nodes} \]

stored again cache-oblivious