CMPT231

Lecture 6: ch18

B-Trees and Midterm Review

2 Corinthians 5:17-19 (NASB)

Therefore if anyone is **in Christ**, he is a **new creature**; the **old** things passed away; behold, **new** things have come.

Now all these things are **from God**,
who **reconciled** us to Himself through Christ
and gave us the **ministry of reconciliation**,

namely, that God was in **Christ**
reconciling the world to Himself,

Outline for today

- **B-Trees**
  - Motivation and **concept**
  - **Search** in $O(t \log_t n)$
  - **Insert** in $O(t \log_t n)$
  - **Delete** in $O(t \log_t n)$
  - Application to **filesystems**
- Midterm **review** (lec1-5, ch1-12 x9)

Balancing search trees

- Complexity of most operations depends on **height**
  - Search, insert, delete
  - **Worst** case: tree becomes a **linked list**
- One approach: regular **rotations**: new root for subtree
  - **Red-black** trees (ch13)
    - Levels alternate **colour**: (max path) ≤ 2x (min path)
  - **AVL** trees: rotate after each insert/delete
  - **Splay** trees: on each search/insert/delete,
    - Rotate node to **root** and rebalance

Spinning-disk storage

- **Seek**: move head to **track**, wait for **sector** *(slow)*
- **Throughput**: read from consecutive sectors *(fast)*

!\[Hard disk, CHS\](static/img/Fig-18-2.svg)

- Lots of **small** iops (I/O ops/sec) are bad
  - So **buffer** and do I/O in larger pages at a time (~16KB)
- **Seek times**: 15ms (laptop), 10ms (desktop), 4ms (server)
  - **Rotational** latency: 5.5ms (laptop), 3ms (server)
- Typical **SSD** seek: 30ns

Trees for disk storage

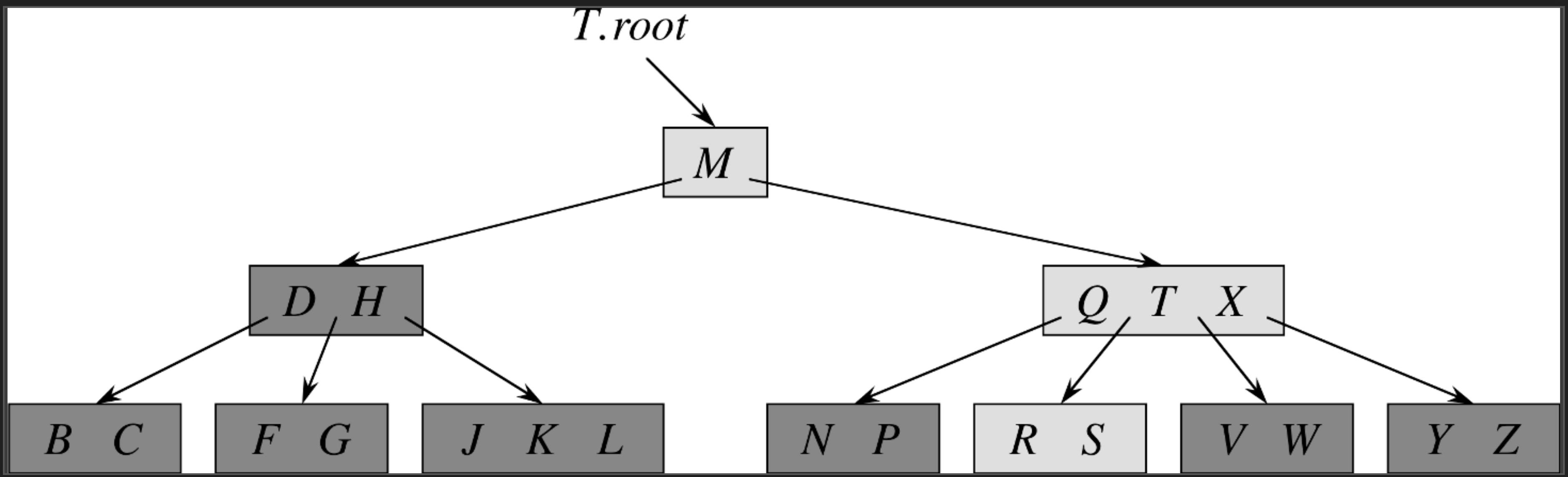
- To edit data on disk:
  - **Read** page from disk into RAM,
  - **Modify** page in-place in RAM, then
  - **Write** page back to disk
- **Disk** operations are very **slow**
- RAM can only store a **limited** number of pages at a time
- **Tree-based** disk filesystem: 1 node = 1 page
  - Want a **low**, bushy tree with large **degree**
  - Generalise **BST** to degree t

B-trees

- In a B-tree of **min-degree** t, every **node** k has:
  - n_k **keys** in sorted order (t-1 $\leq n_k \leq$ 2t-1)
  - $n_k + 1$ **child links**, interleaved between the keys
- **Degree** of each node is between t and 2t
  - Also may be categorised by (Knuth) **order** = 2t
- All **leaves** are at same depth h
- In terms of t and h, what is **min** num of keys stored? **Max**?
- Variants: B+-tree: **payload** stored only in **leaves**
- Variants: B$\ast$-tree: 2t-1 $\leq n_k \leq$ 3t-1

t=2 B-tree (2-3-4 tree)

(**Red-black** tree is a special case of 2-3-4 tree)



(In **B+**-tree, pointers to **data** go in leaf nodes)

Outline for today

- B-Trees
  - Motivation and concept
  - **Search in** $O(t \log_t n)$
  - **Insert in** $O(t \log_t n)$
  - Delete in $O(t \log_t n)$
  - Application to filesystems
- Midterm review (lec1-5, ch1-12 x9)

B-tree operations

- **Search tree** interface: search, insert, delete
- Assess not only **computational** complexity, but also
  - **Disk** accesses (read/write), in terms of n and t
- Keep **root** in RAM (**write** to disk if modified)
  - Other nodes need to be **read** in from disk
- Constraining **degree** (t .. $2t$) keeps tree **balanced**

B-tree search

```
def search( node, key ):

  # Linear search through keys
  for (i = 1; (i <= node.size) and (key > node.keys[i]); i++)

  if ((i <= node.size) and (key == node.keys[i])):
    return (node, i)


    # found it!

  if (node.isLeaf()):
    return None
```

- **Tail** recursion can easily be changed to **loop**
- **Compute** (worst-case): O(th) = $O(t \log_t n)$
- **Disk** accesses (worst-case): O(h) = $O(\log_t n)$

B-tree insert

- As in BST, **search** (down to leaf node)
- As we go, **split** full nodes (2t-1 keys) to ensure free space
  - Split: make **two** nodes with t-1 keys each
  - Promote **median** key up a level
- **Preemptive** split: before we have problems
- Once we reach **leaf** node, we have space to insert

B-tree insert: example

- (a) **initial**: *t=3*
- (b) **non-full** leaf *ACDE*
- (c) full leaf *RSTUV*: **split**
- (d) **split** root *GMPTX*
- (e) **split** node *ABCDE*

`![B-tree insertion example](static/img/Fig-18-7.svg)`

Outline for today

- B-Trees
  - Motivation and concept
  - Search in $O(t \log_t n)$
  - Insert in $O(t \log_t n)$
  - **Delete in $O(t \log_t n)$**
  - **Application to filesystems**
- Midterm review (lec1-5, ch1-12 x9)

B-tree delete

Descend tree, ensuring each node has ≥ t keys

1. If key is **here**, and we're a **leaf**: just delete key
2. If key is **here**, and we're **not** a leaf:
    - (a) If **left** child has ≥ t keys, replace key w/**predecessor**
    - (b) If **right** child has ≥ t keys, replace key w/**successor**
    - (c) Else, **merge** left+right children, and delete key
3. If key is **not** here (and we're not a leaf):

Delete: example

- (a) **initial**: *t=3*
- (b) node *CGM* ≥ *t*, leaf *DEF* ≥ *t*
- (c) key in **internal** node *CGM*: use **predecessor** *L*
- (d) key in **internal**

![B-tree deletion, pt1]
(static/img/Fig-18-8-L.svg)

Delete example (t=3)

`![B-tree deletion, step d](static/img/Fig-18-8-L-d.png)`
`![B-tree deletion, pt2](static/img/Fig-18-8-R.svg)`

B-tree summary

- **Generalisation** of BST, but:
  - All **leaves** are at same **height** ($h = \Theta(\log_t n) = \Theta(\lg n)$)
  - **Degree** of each node is between t and 2t
- **Operations**:
  - **Create**: CPU $O(1)$, disk $O(1)$
  - **Search**/insert/delete: CPU $O(th)$, disk $O(h)$
- When **modifying** tree, as we walk down tree, ensure **degree** of each node stays between t and 2t
  - i.e., number of **keys** stored is between t-1 and 2t-1

Using B-tree in filesystems

- Filesystems store: **files**, **directories**, and **metadata**
  - e.g., name, owner, permissions, modification time)
- File contents are stored in 1 or more **extents** on disk
  - i.e., Logical Block Addresses (LBA) interpretable by HDD
- B-trees can be used for **lookup tables**:
  - **Inode** table: metadata for each object
    - Indexed by inode, unique to each object
  - **Directory** tables: list files in a directory
    - Map filenames (string) to inodes

Filesystems using B-trees

- **NTFS** indexes (i.e., inode tables)
- Mac **HFS** catalog records (i.e., inode tables) use **B+-trees**
- Linux **ext3/4** directory indexes use **Htree** hash tables
  - **Hash** filenames for fast lookup
- Linux **btrfs** (“B-tree filesystem”) uses them everywhere:
  - **Directory** index (with hashed filenames)
  - **Extent** tree (payload is either LBAs or actual data)
  - **Log** tree (journal)

Outline for today

- B-Trees
  - Motivation and concept
  - Search in $O(t \log_t n)$
  - Insert in $O(t \log_t n)$
  - Delete in $O(t \log_t n)$
  - Application to filesystems
- **Midterm review (lec1-5, ch1-12 x9)**

Midterm review

- Tue 25 Oct, **13:10-14:30** (80min)
  - **60 pts**, estimate 1min/pt
- Open paper **book**, open paper **notes**
- No **electronic** devices: computers, phones, etc.
  - Phone should be **off**/mute and in **pocket**/bag
- Bring pen/pencil and **blank** paper to write on
- TA will invigilate; he **cannot** answer questions on content
  - If you feel a question is **ambiguous**, write your interpretation on your exam sheet and answer accordingly

Lecture 1: ch1-3

- **Insertion** sort and its **analysis**
- Discrete **math** review
  - **Logic** and proofs
  - Monotonicity, limits, iterated functions
  - **Fibonacci** sequence and golden ratio
  - Factorials and **Stirling's** approximation
- **Asymptotic** notation: Θ, O, Ω, o, ω
  - **Proving** asymptotic bounds

Lecture 2: ch4-5

- **Divide and conquer** (ch4)
  - **Merge** sort and its **analysis**
  - Recursion trees + proof by **induction**
  - Maximum **subarray**
  - Matrix multiply vs **Strassen**‘s method
  - **Master method** of solving recurrences
- **Probabilistic Analysis** (ch5)
  - **Hiring** problem and analysis
  - **Randomised** algorithms and PRNGs

Lecture 3: ch6-7

- **Heapsort** (ch6) :
  - Trees, binary **heaps**, **max-heap** property
  - **Heapify()** function on a node
  - **Heapsort**: build a max-heap, use it for sorting
  - **Priority queue** using max-heap: operations, complexity
- **Quicksort** (ch7) :
  - Regular quicksort with **fixed** (Lomuto) partitioning
  - **Randomised** pivot
  - Analysis of randomised pivot: **expected** time

Lecture 4: ch8,11

- **Linear**-time sorts (ch8) (**assumptions!**)
  - **Decision-tree** model, why comparison sorts are $\Omega(n \lg n)$
  - **Counting** sort: census + move: $\Theta(n+k)$, **stability**
  - **Radix** sort (with r-bit digits): $\Theta(d(n+k))$
  - **Bucket** sort: $\Theta(n)$ **expected** time
- **Hash** tables (ch11):
  - Hash **function**, hash **collisions**, **chaining**
  - **Load factor** α = n/num_buckets
  - **Search** in $\Theta(1+\alpha)$
  - **Hashes**: div, mul, universal hashing

Lecture 5: ch10,12

- **Linked lists** (ch10):
  - Singly/**doubly**-linked, **circular**
- **Stacks** and **queues** (ch10):
  - Operations, implementation with linked-lists
- **Trees** and Binary search trees (**BST**) (ch12):
  - Tree **traversals**: inorder, preorder, postorder
  - **Searching** a BST
  - **Min**/max and **successor**/predecessor
  - **Insert** and **delete**
  - **Randomised** BST
- **Skip lists**

Outline for today

- **B-Trees**
  - Motivation and **concept**
  - **Search** in $O(t \log_t n)$
  - **Insert** in $O(t \log_t n)$
  - **Delete** in $O(t \log_t n)$
  - Application to **filesystems**
- Midterm **review** (lec1-5, ch1-12 x9)