Exam Logistics

Take-home kept in the registrar’s office

- I’ll create repositories like I did for the midterm, including
  - Instruction page
  - Blank question template
  - Not required to typeset (registrar gives out “blue books”, or use your own paper or write on exam, or combination of the above)
    - But must print out and submit your exam inside registrar’s envelope

- Review session:
  - Monday evening from 7:30-8:30pm in TCL 206

- Practice questions:
  - Best tools are homework and questions found in slides
  - Rule of thumb for exam questions: will be representative of things you’ve done on homework or in-class activities (POGIL sessions in Ward Lab)

30-minute Question Review Sessions

- Will be sign-up slots for all day Monday and Tuesday
  - Happy to do 1-off meetings if those don’t work for you
  - Not at all required; don’t need to do 4 questions; use as you’d like
  - Email me which questions you wish to go over so I can have them ready
Many TA hours will happen

• Most TAs were happy to host normal hours over reading period
  ▶ I’ll update the course schedule once I get their finalized hours

• 30-minute slots can be used however you’d like

• I’m happy to add additional office hours throughout the week, on-demand!
  ▶ Let me know in advance and I can advertise a time to the mailing list
B-trees (Ubiquitous and otherwise)
Indexing Overview
- General task & properties

DAM model
- How to analyze external memory algorithms

B-trees
- Operations
- Variants
- Discussion

Why B-trees, why now when the semester is over?
- We’ve studied many algorithms and data structures; here is an example of how we can change our underlying algorithmic model and then apply the same analysis tools we’ve built throughout

Not on the exam; goal is to show you the flexibility and power of what you’ve learned
Big Picture Question: How do you keep data organized?
An Analogy from [Comer 79 CSUR]

Filing cabinet with folders of employee records, alpha-sorted by employee last name

• We often think in terms of keys and values. Here?
  ▸ Keys are the employees’ last names
  ▸ Values are the employee file (held in a folder, one per employee)

• A filing cabinet effectively supports two types of searches
  ▸ Sequential
    ▸ read through every folder in every drawer, in order
  ▸ Random (targeted)
    ▸ use the labels on the drawers & folders to find the single record of interest

In this scenario, the filing cabinet is an **index** of employee records
Indexes organize data

- Random (targeted) searches utilize an index to:
  - Direct our search towards a small part of the total data
  - (Hopefully) speed up our search

Questions

- What operations does an index support?
- How do we quantify index performance?
- Is the data part of the index, or does the index “sit on top of” the data?
  - Filing cabinet storing employee folders (data part of index) vs. library card catalog storing book call numbers (pointer to data stored in index)
Q1: What operations does an index support?

Operations

• Insert(k,v): inserts key-value pair (k,v)
• Delete(k): deletes any pair (k, *)
• PointQuery(k): returns all pairs (k, *)
• RangeQuery(k₁,k₂): returns all pairs (k, *), k₁ ≤ k ≤ k₂

In short, indexes support the dictionary interface.

• Often used for very large data sets.
Q2: How to we quantify index performance?

Algorithmically, we can use the DAM model:
- Useful model in scenarios when data is too big for memory
  - Data is transferred in blocks between RAM and disk.
- Premise: the number of block transfers dominates the running time.
  - Searching through a given block is “free” (once block is in-memory)

Goal: Minimize # of I/Os
- Performance bounds are parameterized by block size $B$, memory size $M$, data size $N$.

[Aggarwal+Vitter ’88]
Analyze worst-case costs by counting I/Os

- **B**: unit of transfer
  - B-tree node size
- **M**: amount of main memory
  - We can cache M/B nodes in memory at once
- **N**: size of our data
  - We’re not worried about disk space, we use N to describe our tree

- We will think about the tree shape (node size, height, fanout), then describe each operation’s cost in terms of the DAM model
The B-tree
B-trees store records

- Records are key-value pairs
- We assume that keys are
  - Unique (to simplify analysis)
  - Ordered
Rules for our B-trees

• **B-ary tree**
  ‣ Internal nodes have between $d$ and $2d$ keys called *pivots*
    ‣ This means nodes must always be half full!
    ‣ At least $d+1$ pointers to children (one more pointer than pivot key)

• If an operation would cause a violation of one of these invariants, must rebalance the tree!

• *Note: our B-tree’s internal nodes do not* store records
  ‣ Option 1: Store (key, value) pairs in leaves
  ‣ Option 2: Store (key, pointer to value) in leaves
Several B-tree variants

- We will describe a “B?!+-tree” here, noting features of specific variants as they come up

Popular Variants of B-trees

- B-tree: more-or-less what we’ll describe here
- B+-tree: B-tree where leaves form a linked list
- B*-tree: B-tree where nodes always 2/3 full
B-tree: standard DAM dictionary

B-ary search tree

What does B Stand for?

\[ \geq \text{half full} \]

\[ O(\log_B N) \]

Summary

<table>
<thead>
<tr>
<th></th>
<th>Point Query</th>
<th>Insert</th>
<th>Delete</th>
<th>Range Query</th>
</tr>
</thead>
<tbody>
<tr>
<td>B-tree</td>
<td>( O(\log_B N) )</td>
<td>( O(\log_B N) )</td>
<td>( O(\log_B N) )</td>
<td>( O\left(\log_B N + \frac{K}{B}\right))</td>
</tr>
</tbody>
</table>
B-tree Point Queries
Steps

• Starting at the root, find the first **pivot key** that is larger than your **search key**, and follow the pointer on its left
  ▸ If there are no pivot keys larger than your search key, follow the last pointer

• Repeat until you arrive at a leaf node

• Search the leaf node (ordered list) for your target key

• Return the key-value pair (if found), or **NONE**

---

This work is done during an insert (need to find place where new key-value pair belongs), so we will walk through this then.
Cost

• **How many nodes must be read/written in a search?**
  ‣ We read the root node to search the pivot keys
  ‣ We recurse on the subtree

• **Total cost of a search: \( O(h) \)**
  ‣ Recall \( h = O(\log_B N) \)
B-tree Insertions
B-tree Insert

Steps

• Find the leaf node where your key-value pair belongs (point query)
• Insert your key-value pair into that leaf
B-tree: standard DAM dictionary

B-ary search tree

Summary

<table>
<thead>
<tr>
<th></th>
<th>Point Query</th>
<th>Insert</th>
<th>Delete</th>
<th>Range Query</th>
</tr>
</thead>
<tbody>
<tr>
<td>B-tree</td>
<td>$O(\log_B N)$</td>
<td>$O(\log_B N)$</td>
<td>$O(\log_B N)$</td>
<td>$O\left(\log_B N + \frac{K}{B}\right)$</td>
</tr>
</tbody>
</table>
B-tree: standard DAM dictionary

B-ary search tree

Summary

<table>
<thead>
<tr>
<th>Summary</th>
<th>Point Query</th>
<th>Insert</th>
<th>Delete</th>
<th>Range Query</th>
</tr>
</thead>
<tbody>
<tr>
<td>B-tree</td>
<td>$O(\log_B N)$</td>
<td>$O(\log_B N)$</td>
<td>$O(\log_B N)$</td>
<td>$O\left(\log_B N + \frac{K}{B}\right)$</td>
</tr>
</tbody>
</table>
### B-tree: standard DAM dictionary

#### B-ary search tree

![B-tree diagram](image)

#### Summary

<table>
<thead>
<tr>
<th>Operation</th>
<th>B-tree</th>
</tr>
</thead>
<tbody>
<tr>
<td>Point Query</td>
<td>$O(\log_B N)$</td>
</tr>
<tr>
<td>Insert</td>
<td>$O(\log_B N)$</td>
</tr>
<tr>
<td>Delete</td>
<td>$O(\log_B N)$</td>
</tr>
<tr>
<td>Range Query</td>
<td>$O(\log_B N + \frac{K}{B})$</td>
</tr>
</tbody>
</table>
B-tree: standard DAM dictionary

B-ary search tree

Summary

<table>
<thead>
<tr>
<th></th>
<th>Point Query</th>
<th>Insert</th>
<th>Delete</th>
<th>Range Query</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>B-tree</strong></td>
<td>$O(\log_B N)$</td>
<td>$O(\log_B N)$</td>
<td>$O(\log_B N)$</td>
<td>$O\left(\log_B N + \frac{K}{B}\right)$</td>
</tr>
</tbody>
</table>
B-tree: standard DAM dictionary

B-ary search tree

Summary

<table>
<thead>
<tr>
<th></th>
<th>Point Query</th>
<th>Insert</th>
<th>Delete</th>
<th>Range Query</th>
</tr>
</thead>
<tbody>
<tr>
<td>B-tree</td>
<td>$O(\log_B N)$</td>
<td>$O(\log_B N)$</td>
<td>$O(\log_B N)$</td>
<td>$O\left(\log_B N + \frac{K}{B}\right)$</td>
</tr>
</tbody>
</table>
B-tree: standard DAM dictionary

B-ary search tree

Summary

<table>
<thead>
<tr>
<th></th>
<th>Point Query</th>
<th>Insert</th>
<th>Delete</th>
<th>Range Query</th>
</tr>
</thead>
<tbody>
<tr>
<td>B-tree</td>
<td>$O(\log_B N)$</td>
<td>$O(\log_B N)$</td>
<td>$O(\log_B N)$</td>
<td>$O\left(\log_B N + \frac{K}{B}\right)$</td>
</tr>
</tbody>
</table>
B-tree: standard DAM dictionary

B-ary search tree

Summary

\begin{tabular}{|c|c|c|c|}
\hline
 & Point Query & Insert & Delete & Range Query \\
\hline
B-tree & $O(\log_B N)$ & $O(\log_B N)$ & $O(\log_B N)$ & $O\left(\log_B N + \frac{K}{B}\right)$ \\
\hline
\end{tabular}
**B-tree: standard DAM dictionary**

**B-ary search tree**

Summary

<table>
<thead>
<tr>
<th></th>
<th>Point Query</th>
<th>Insert</th>
<th>Delete</th>
<th>Range Query</th>
</tr>
</thead>
<tbody>
<tr>
<td>B-tree</td>
<td>$O(\log_B N)$</td>
<td>$O(\log_B N)$</td>
<td>$O(\log_B N)$</td>
<td>$O \left( \log_B N + \frac{K}{B} \right)$</td>
</tr>
</tbody>
</table>
B-tree: standard DAM dictionary

B-ary search tree

Summary

<table>
<thead>
<tr>
<th></th>
<th>Point Query</th>
<th>Insert</th>
<th>Delete</th>
<th>Range Query</th>
</tr>
</thead>
<tbody>
<tr>
<td>B-tree</td>
<td>$O(\log_B N)$</td>
<td>$O(\log_B N)$</td>
<td>$O(\log_B N)$</td>
<td>$O\left(\log_B N + \frac{K}{B}\right)$</td>
</tr>
</tbody>
</table>
B-tree: standard DAM dictionary

B-ary search tree

Summary

<table>
<thead>
<tr>
<th></th>
<th>Point Query</th>
<th>Insert</th>
<th>Delete</th>
<th>Range Query</th>
</tr>
</thead>
<tbody>
<tr>
<td>B-tree</td>
<td>$O(\log_B N)$</td>
<td>$O(\log_B N)$</td>
<td>$O(\log_B N)$</td>
<td>$O\left(\log_B N + \frac{K}{B}\right)$</td>
</tr>
</tbody>
</table>
B-tree: standard DAM dictionary

B-ary search tree

Summary

<table>
<thead>
<tr>
<th></th>
<th>Point Query</th>
<th>Insert</th>
<th>Delete</th>
<th>Range Query</th>
</tr>
</thead>
<tbody>
<tr>
<td>B-tree</td>
<td>$O(\log_B N)$</td>
<td>$O(\log_B N)$</td>
<td>$O(\log_B N)$</td>
<td>$O\left(\log_B N + \frac{K}{B}\right)$</td>
</tr>
</tbody>
</table>
B-tree: standard DAM dictionary

B-ary search tree

Summary

<table>
<thead>
<tr>
<th></th>
<th>Point Query</th>
<th>Insert</th>
<th>Delete</th>
<th>Range Query</th>
</tr>
</thead>
<tbody>
<tr>
<td>B-tree</td>
<td>$O(\log_B N)$</td>
<td>$O(\log_B N)$</td>
<td>$O(\log_B N)$</td>
<td>$O\left(\log_B N + \frac{K}{B}\right)$</td>
</tr>
</tbody>
</table>
B-tree: standard DAM dictionary

B-ary search tree

No room! Need to split the node.

Summary

<table>
<thead>
<tr>
<th></th>
<th>Point Query</th>
<th>Insert</th>
<th>Delete</th>
<th>Range Query</th>
</tr>
</thead>
<tbody>
<tr>
<td>B-tree</td>
<td>$O(\log_B N)$</td>
<td>$O(\log_B N)$</td>
<td>$O(\log_B N)$</td>
<td>$O\left(\log_B N + \frac{K}{B}\right)$</td>
</tr>
</tbody>
</table>
Splitting a B-tree node

Steps

- Sort all $2d+1$ keys ($2d + \text{new key that causes overflow}$)
- Make new node with first $d$ keys
- Make new node with last $d$ keys
- Move middle key as a pivot of the parent
- Add pointers to new children
- Recurse up the tree if necessary (rare)
B-tree: standard DAM dictionary

B-ary search tree

Summary

<table>
<thead>
<tr>
<th></th>
<th>Point Query</th>
<th>Insert</th>
<th>Delete</th>
<th>Range Query</th>
</tr>
</thead>
<tbody>
<tr>
<td>B-tree</td>
<td>$O(\log_B N)$</td>
<td>$O(\log_B N)$</td>
<td>$O(\log_B N)$</td>
<td>$O\left(\log_B N + \frac{K}{B}\right)$</td>
</tr>
</tbody>
</table>
B-tree: standard DAM dictionary

B-ary search tree

Summary

<table>
<thead>
<tr>
<th></th>
<th>Point Query</th>
<th>Insert</th>
<th>Delete</th>
<th>Range Query</th>
</tr>
</thead>
<tbody>
<tr>
<td>B-tree</td>
<td>$O(\log_B N)$</td>
<td>$O(\log_B N)$</td>
<td>$O(\log_B N)$</td>
<td>$O\left(\log_B N + \frac{K}{B}\right)$</td>
</tr>
</tbody>
</table>
B-tree: standard DAM dictionary

B-ary search tree

Summary

<table>
<thead>
<tr>
<th></th>
<th>Point Query</th>
<th>Insert</th>
<th>Delete</th>
<th>Range Query</th>
</tr>
</thead>
<tbody>
<tr>
<td>B-tree</td>
<td>$O(\log_B N)$</td>
<td>$O(\log_B N)$</td>
<td>$O(\log_B N)$</td>
<td>$O(\log_B N + \frac{K}{B})$</td>
</tr>
</tbody>
</table>
B-tree: standard DAM dictionary

B-ary search tree

<table>
<thead>
<tr>
<th>Summary</th>
<th>Point Query</th>
<th>Insert</th>
<th>Delete</th>
<th>Range Query</th>
</tr>
</thead>
<tbody>
<tr>
<td>B-tree</td>
<td>$O(\log_B N)$</td>
<td>$O(\log_B N)$</td>
<td>$O(\log_B N)$</td>
<td>$O\left(\log_B N + \frac{K}{B}\right)$</td>
</tr>
</tbody>
</table>
B-tree: standard DAM dictionary

B-ary search tree

Summary

<table>
<thead>
<tr>
<th></th>
<th>Point Query</th>
<th>Insert</th>
<th>Delete</th>
<th>Range Query</th>
</tr>
</thead>
<tbody>
<tr>
<td>B-tree</td>
<td>$O(\log_B N)$</td>
<td>$O(\log_B N)$</td>
<td>$O(\log_B N)$</td>
<td>$O\left(\log_B N + \frac{K}{B}\right)$</td>
</tr>
</tbody>
</table>
Splitting a B-tree node

Cost

• How many nodes must be read/written in a local split?
  ▸ We read the node being split
  ▸ We write the old node and the new node (first d keys, last d keys)
  ▸ We read/write the parent node

• What if we overflow the parent?
  ▸ If we recurse, we already read the parent, so we repeat the same steps one level above

• Total cost of an insert: O(h)
  ▸ Reads: O(h)
  ▸ Writes: O(2h)
B-tree Range Queries
B-tree Range Query

(Range query: point query + successor

Steps

• Find the leaf node where the first key-value pair belongs (point query)
• Read all key-value pairs from that node that are part of your range
• Consult your parent to find its next child pointer
• Read all key-value pairs from that node that are part of your range
• Loop
B-tree: standard DAM dictionary

B-ary search tree

Summary

<table>
<thead>
<tr>
<th>Summary</th>
<th>Point Query</th>
<th>Insert</th>
<th>Delete</th>
<th>Range Query</th>
</tr>
</thead>
<tbody>
<tr>
<td>B-tree</td>
<td>$O(\log_B N)$</td>
<td>$O(\log_B N)$</td>
<td>$O(\log_B N)$</td>
<td>$O\left(\log_B N + \frac{K}{B}\right)$</td>
</tr>
</tbody>
</table>
B-tree Deletes
B-tree Deletions

Steps

• Search for the leaf containing the target key-value pair (point query)
• Remove the element from the leaf (if present)
• If the size of the node drops below $d$, merge with a neighbor
  ▶ Remove extra pivot key and pointer from parent (the pointer to the node that is being deleted as part of the merge)
  ▶ Merge contents of nodes
  ▶ Write parent and merged node
  ▶ If the parent size dropped below $d$, recurse upwards
B-tree: standard DAM dictionary

B-ary search tree

Summary

<table>
<thead>
<tr>
<th></th>
<th>Point Query</th>
<th>Insert</th>
<th>Delete</th>
<th>Range Query</th>
</tr>
</thead>
<tbody>
<tr>
<td>B-tree</td>
<td>$O(\log_B N)$</td>
<td>$O(\log_B N)$</td>
<td>$O(\log_B N)$</td>
<td>$O\left(\log_B N + \frac{K}{B}\right)$</td>
</tr>
</tbody>
</table>
B-tree: standard DAM dictionary

B-ary search tree

Summary

<table>
<thead>
<tr>
<th></th>
<th>Point Query</th>
<th>Insert</th>
<th>Delete</th>
<th>Range Query</th>
</tr>
</thead>
<tbody>
<tr>
<td>B-tree</td>
<td>$O(\log_B N)$</td>
<td>$O(\log_B N)$</td>
<td>$O(\log_B N)$</td>
<td>$O\left(\log_B N + \frac{K}{B}\right)$</td>
</tr>
</tbody>
</table>
B-tree: standard DAM dictionary

B-ary search tree

Summary

<table>
<thead>
<tr>
<th>Summary</th>
<th>Point Query</th>
<th>Insert</th>
<th>Delete</th>
<th>Range Query</th>
</tr>
</thead>
<tbody>
<tr>
<td>B-tree</td>
<td>$O(\log_B N)$</td>
<td>$O(\log_B N)$</td>
<td>$O(\log_B N)$</td>
<td>$O \left( \log_B N + \frac{K}{B} \right)$</td>
</tr>
</tbody>
</table>
**B-tree: standard DAM dictionary**

**B-ary search tree**

Too small! Need to merge

Summary

<table>
<thead>
<tr>
<th>Summary</th>
<th>Point Query</th>
<th>Insert</th>
<th>Delete</th>
<th>Range Query</th>
</tr>
</thead>
<tbody>
<tr>
<td>B-tree</td>
<td>$O(\log_B N)$</td>
<td>$O(\log_B N)$</td>
<td>$O(\log_B N)$</td>
<td>$O\left(\log_B N + \frac{K}{B}\right)$</td>
</tr>
</tbody>
</table>
B-tree: standard DAM dictionary

B-ary search tree

Summary

<table>
<thead>
<tr>
<th></th>
<th>Point Query</th>
<th>Insert</th>
<th>Delete</th>
<th>Range Query</th>
</tr>
</thead>
<tbody>
<tr>
<td>B-tree</td>
<td>$O(\log_B N)$</td>
<td>$O(\log_B N)$</td>
<td>$O(\log_B N)$</td>
<td>$O\left(\log_B N + \frac{K}{B}\right)$</td>
</tr>
</tbody>
</table>
B-tree: standard DAM dictionary

B-ary search tree

Summary

<table>
<thead>
<tr>
<th></th>
<th>Point Query</th>
<th>Insert</th>
<th>Delete</th>
<th>Range Query</th>
</tr>
</thead>
<tbody>
<tr>
<td>B-tree</td>
<td>$O(\log_B N)$</td>
<td>$O(\log_B N)$</td>
<td>$O(\log_B N)$</td>
<td>$O(\log_B N + \frac{K}{B})$</td>
</tr>
</tbody>
</table>
Summary

• B-trees are the de-facto search structure for external memory applications
• Variants exist to tune utilization and range scan performance, but the idea is the same
• We can analyze performance using the DAM model

Other discussions

• Concurrent access - how to lock the tree?
  ▸ Hand-over-hand locking for queries
  ▸ Reservations or top-down splitting
• How to choose the node size (B)?
  ▸ Must balance competing goals:
    ▸ Small B minimizes write amplification (each update requires writing whole node)
    ▸ Large B minimizes fragmentation (more data read per seek)
B-trees because they are widely used, but they also serve as a starting point to discuss more recent advances in trees

- Log structured merge trees
- B\textsuperscript{e}-trees

The above trees employ *write optimization*

- Better I/O performance for writes
- Not asymptotically worse off for reads

If you like this type of analysis (the intersection of asymptotic analysis and system optimization)

- Several electives!
  - Applied algorithms
  - Storage Systems
  - Parallel Processing // Distributed Systems
We’ve covered a lot this semester

- **My first time teaching 256, so I’ve learned a lot**
  - I hope to learn what was most helpful for you
- **Covered important topics that will help you think about, formally define, and quantify problems and their solutions**
  - Asymptotic analysis
  - Graphs: traversals, algorithms, and applications
  - Greedy algorithms & proving their optimality
  - Divide and conquer // Recurrences
  - Dynamic programming
  - Network flow & problem reductions
  - Intractability: NP & NP hardness & more problem reductions
  - Randomized algorithms and analysis
What’s Next?

CS256 Opens up several doors

• Prerequisite to both theory and “applications” electives
• Prerequisite to Theory of Computation
• There isn’t anyone in this room I wouldn’t recommend for research in CS (if that is what you actually want…)
  ▶ Theorists in CS faculty include…
    ▶ Sam McCauley
    ▶ Shikha Singh
    ▶ Aaron Williams
  ▶ Intersection of Theory and XXXXX
    ▶ Data structures/indexing/filters (Bill & Sam & Shikha)
    ▶ Many PL problems (Dan & Steve)
    ▶ Distributed Systems (Jeannie)
    ▶ Algorithmic Game Theory (Shikha)

• Summer research, theses, and (sometimes) RAs
• REU programs are available in other places too!
http://cs.williams.edu/~jannen/teaching/cs-prereqs.svg