Eric Andrews, Christopher Chapman

CPS377: Database Systems

Professor Justin Brody

Final Project: Empirically Benchmarking Tree-based and Hash-based Database Implementations

GitHub: `https://github.com/CWMChapman/DBMS-Implementation`

Presentation Preferred Date: December 1, 2-5pm

## Milestone 2 Report

### Page Manager

*Done*

Since the last milestone, we have added a `remRecord` function that take in an `rid` and deletes the associated record. If that record page is empty after performing the operation, it is freed.

*Planned—For Final Submission*

Once the data structures are fully implemented and polished, if time permits we will add some simulation of buffer pages. Right now the page manager naïvely assumes that every read and write must go to disk. Further granularity in the data might also be nice—for example, which reads and writes are the cost of traversing the data structure and which are the cost of retrieving the data from memory.

Additionally, the way in which the page manager "stores" on disk is currently very naïve: it tracks `curRecordPage`, and every time a record is added it puts it on that page. When the page is full, it "writes" to disk and makes a new page. This works great so long as you only ever insert items. For the final submission, we will work out a somewhat more sophisticated and efficient solution; if time permits, we may also introduce clustered and unclustered data.

The `remRecord` function still leaves much to be desired; it is currently very memory-inefficient, as it does not allow for empty slots to be refilled. Once removes are implemented on the index side, `remRecord` will be updated to be more sophisticated; right now its only purpose is to provide some interface for removes to access.

### B+ Tree

*Done*

- Insert is now fully working. There were a couple of bugs around splitting buckets that took far longer than anticipated to squash.
- A simple equality search has been implemented.
- Insert now calls `getPage` when appropriate, allowing rudimentary benchmarking
- A `checkTree` function has been added, which recursively scans the tree and ensures that it is valid. It checks that each node is in order, that its values are contained within the parameters of its parent,

and that the leaf nodes contain all values in the tree in order. Right now, it assumes that the tree will contain the complete list of integers 0 through $n$, where $n$ is the number of items in the tree.

*Planned—For Final Submission*

- Remove
- Range search
- `checkTree` should update its assumptions to test `remove`—it will have to take in an array of the expected values in the tree (or perhaps an array of removed values to ignore) rather than generating that implicitly.
- Benchmarking
- For simplicity, search and insert currently utilize a linear search within nodes, rather than the more efficient binary search; they should be updated. This is very low priority, as the two are functionally identical and do not affect the number of page I/Os, which is what we're actually benchmarking.

**Linear Hash**

*Done*

- Insert and lookup with overflow buckets work now! This took me a lot of time to do actually... there were lots of bugs primarily because of how complex the hashing scheme became. I've tested insert and lookup pretty rigorously and it works well!
- Split buckets doesn't work quite yet. It successfully doubles the number of bucket. Which was somewhat difficult to do in c since it's not so simple to resize an array. I'm currently trying to rehash (reinsert) all of the entries into the bucket being split based on the new hash function. This is quite challenging to implement. There are lot of things to think about like which hash function to use for which entries based on which bucket is next to be split (i.e. which level we are on in the linear hashing scheme).

*Planned—For Final Submission*

- Split buckets and delete entries based on key.
- Remove
- Benchmarking the setup