P2 - B+ Tree

Deadline: Sunday, March 10; 11:59 PM

Accept Assignment: https://classroom.github.com/a/gd95nzgc

Submit Assignment: https://autolab.cse.buffalo.edu/courses/cse410-s24/assessments/P2-B-Trees

In this assignment, you will implement an on-disk B+Tree.

This assignment is intended to: - Give you experience building a paged, on-disk data structure - Give you experience in enforcing data structure constraints - Give you experience working with an existing codebase

You should expect to spend approximately 20-30 hours on this assignment. Plan accordingly.

To complete this assignment, you should:

1. Accept this assignment through GitHub Classroom.
2. Modify the file `src/bplus_tree.rs`, implementing the functions labeled `todo!()`.
3. Commit your changes and push them to Github.
4. Go to Autolab, select your repository, acknowledge the course AI Policy, and click Submit.

You may repeat steps 2-4 as many times as desired. You may also modify any of the files in the `page` module.

Overview

In this assignment, you will implement key parts of an on-disk B+Tree.

Documentation

You may find the following documentation useful:

- The Rust Book
- std::fs::File

- Run `cargo doc --open`

The following utility methods are provided, and may be useful

`bplus_tree::BPlusTree::init(path)`

Initialize a fresh BPlusTree backed by the file at the specified path.

`bplus_tree::BPlusTree::open(path)`

Open an existing BPlusTree backed by the file at the specified path.

`bplus_tree::BPlusTree::get_page(&self, idx)`

Retrieve the page at the specified index. The type of the page read is determined by Rust's typesystem. Both of the following approaches work: `let page: DirectoryPage = self.get_page(idx); let page = self.get_page::<DirectoryPage>(idx);`

`bplus_tree::BPlusTree::put_page(&self, idx, &page)`

Write the provided page to disk at the provided index.

`bplus_tree::BPlusTree::check_tree(&self)`

Sanity check the contents of the tree. If any standard assumptions are not met, this method returns Ok(Some(err_msg))

`bplus_tree::BPlusTree::print_tree(&self)`

Print out the tree to standard out

The `page` module

The `page` module (`src/page/mod.rs`) provides functionality for reading and writing different types of pages. Each implementation of the page trait (`DirectoryPage`, `LeafPage`, `FreePage`, and `MetadataPage`) provides functionality for manipulating the page. See the project documentation (`cargo doc --open`) for more details.

General File Structure

We're going to assume that page 0 contains a `MetadataPage`. After calling `::init()` or `::open()`, the metadata page contents will be available as the `.meta` field of the returned

tree.

Note that you *must* manually write changes to the metadata page back to disk. There is a convenience method for this: `put_meta()`

Note the contents of the `MetadataPage` object. Test cases assume the following:

- `next_free_page` : A pointer to the first free page or NULL_IDX otherwise.
- `root_page` : A pointer to the root directory page.
- `data_head` : A pointer to the first leaf page.
- `data_tail` : A pointer to the last leaf page.
- `pages_allocated` : The number of pages allocated in the file (including the metadata page).
- `depth` : The number of levels of directory pages in the file.

There must always be at least one directory page and one leaf page, even in an empty file.

Finally, recall the B+Tree constraint: A Directory/Leaf page must be at least 50% full at all times. The only exceptions to this rule are: - The root directory page may contain fewer than 50% entries, but must contain at least one key. - The root directory page of a depth=1 tree may be completely empty. - If a tree contains only a single leaf page, this leaf page may contain fewer than 50% entries.

The `.is_underfull()` and `.can_allow_stolen_key()` methods on `DirectoryPage` and `LeafPage` can help to enforce these constraints.

Objectives

In this assignment, you will implement four functions:

`bplus_tree::BPlusTree::alloc_page(&self, &page)`

This method should allocate a new page for use by the caller and write the provided page to it. If a previously freed page is available, this should be used first. Otherwise, `alloc_page` should write the page to the end of the file.

`alloc_page` should ensure that the metadata page is appropriately updated: - `free_page` should

Complexity: - Memory: O(1) - IO: O(1)

`bplus_tree::BPlusTree::free_page(&self, &page)`

This method should release a page after use. The page contents should be overwritten with a `FreePage` and the page should be made available for use by a subsequent call to `alloc_page` (e.g., by marking it in the `MetadataPage`'s `next_free_page` field).

Complexity: - Memory: O(1) - IO: O(1)

`bplus_tree::BPlusTree::put(&self, key, value)`

This method should insert a key/value pair into the tree. Subsequent calls to `get(key)` should return `value`. If `key` is already present, the prior value should be overwritten.

The write should persist restarts; if the same file is later re-opened `get(key)` should still return `value`.

Note: Recall that if a leaf page splits, it may also trigger splits in the ancestors.

Complexity: - Memory: O(log_K(N)) - IO: O(log_K(N)) reads, amortized O(1) writes (worst case O(log_K(N)))

`bplus_tree::BPlusTree::delete(&self, key)`

This method should remove a key from the tree. Subsequent calls to `get(key)` should return `None`.

The delete should persist restarts.

Recall from the notes above that no page (leaf or directory) should be less than 50% full (see `is_underfull()`). An underfull page can be addressed in one of the following ways: - 'Stealing' a record from the preceding or following sibling page (i.e., page with the same parent). - 'Merging' with the preceding or following sibling page.

Since stealing is possible for any page at over 50% capacity, and merging is possible for any two pages at 50% capacity or below, one of the two options will always be possible.

Complexity: - Memory: O(log_K(N)) - IO: O(log_K(N)) reads, amortized O(1) writes (worst case O(log_K(N)))

Strategy

Note the `DIR_KEY_COUNT` and `LEAF_RECORD_COUNT` constants defined in `src/page/dir_page.rs` and `src/page/leaf_page.rs` respectively. In the template file, these are set as high as possible. However, for the purpose of debugging, you may find it convenient to set them to lower values (e.g., `4`).

A compile-time assertion defined will not allow you to set these values to be greater than the size of a single page.

You are encouraged to subdivide the problem into 3 phases. The provided test cases are designed accordingly:

1. Implement a page allocator (`alloc_page` , `free_page`) and pass the first provided test case.
2. Implement the `put` method and pass the second provided test case.
3. Implement the `delete` method and pass the third provided test case.

The `put` and `delete` methods also benefit from being broken down into smaller cases. You may find it convenient to solve these cases one at a time. The `todo!()` macro can be very helpful here, allowing you to test implementations of one case at a time, as you develop them.

Put

1. There is enough space on the leaf page.
2. The leaf page needs to split, but its parent directory page has enough space.
3. The leaf page and its immediate parent need to split, and the immediate parent is the root page.
4. The leaf page and its immediate parent need to split, and the immediate parent is not the root page.
5. The leaf page and one or more of its ancestors need to split.

Get

1. The leaf page is not underfull after deletion.
2. The leaf page is underfull, but one of its siblings can be stolen from.
3. The leaf page is underfull, neither of its siblings can be stolen from, but the immediate parent doesn't become underfull after merging the leaf pages.
4. The leaf page is underfull, neither of its siblings can be stolen from, and the immediate parent can steal from one of its siblings.
5. The leaf page is underfull, neither of its siblings can be stolen from, the immediate parent is underfull, and neither of its siblings can be stolen from.
6. The leaf page and one or more of its ancestors need to merge.

Additional Notes

- In addition to `src/bplus_tree.rs`, you may modify any of the existing files in `src/mod`.
- You may modify the structure of the files and page layouts, as long as you pass the provided test cases.
- You may **not** add new crates without permission.