P1 - Binary Search (On Disk)

Deadline: Sunday, Feb 11; 11:59 PM

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

Submit Assignment: https://autolab.cse.buffalo.edu/courses/cse410-s24/assessments/P1-Binary

In this assignment, we will implement binary search using O(1) memory using file handles.

This assignment is intended to: - Familiarize you with Rust and Cargo - Familiarize you with Rust's `File` API, including `Seek` - Familiarize you with working with binary data encodings - Familiarize you with implementing bounded-memory algorithms.

You should expect to spend approximately 10-15 hours on this assignment. Plan accordingly.

To complete this assignment, you should:

1. Accept this assignment through GitHub Classroom.
2. Modify the file `src/data_file.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

Overview

In this assignment, you will be provided with a data file consisting of an arbitrary number of serialized `Record` objects, each consisting of a `key` and a `value`. Each record will have a unique `key`, and records will be stored in ascending sorted order of their `key`.

Your `data_file::DataFile` implementation should be able to: - Open the file - Retrieve the nth record from the file - Perform an O(1)-memory binary search over the file to find a specific key

---

Documentation

You may find the following documentation useful:

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

The following utility methods are provided for your convenience:

`buffer_to_record(buffer)`

Given a buffer, exactly the size of one record, this function will transmute it into a Record object.

Objectives

In this assignment, you will implement three functions:

`DataFile::open(path)`

This method should instantiate a DataFile object using the file at the provided path. Note the four fields of a `DataFile` : * `file` : A `File` reference storing an open, read-only filehandle. * `number_of_records` : The number of records in the file. * `min_key` : The least key of any record in the file (the key of the first record) * `max_key` : The greatest key of any record in the file (the key of the last record)

You should derive the `number_of_records` , `min_key` , and `max_key` attributes directly from the file. The length of the file (in bytes) is given as part of the file's `Metadata` .

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

`data_file.get(idx)`

This method should return the `idx` th record stored in the file. If `idx` is out of bounds, you should panic.

Note the `buffer_to_record` helper function.

Note also the bound on memory.

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

`data_file.find(key)`

If a record with key `key` is present in the file, this method should return it. If a record is not present, this function should return: - The successor of `key` (the record with the next highest key) if one exists - None if `key` has no successor

You may assume that the records in the file are stored in sorted order.

Note the bound on memory.

Complexity: - Runtime: O(log_2(N)) - Memory: O(1) - IO: O(log_2(N))