CS 271 Project 5 – There B-Trees

Instructor: Flannery Currin

Due: November 10th, 2025 by 9:00 AM

1 Learning Goals

B-Trees build on the balancing power of Red-Black Trees but increase the log base in operations' time complexity by allowing nodes to have more children and reduce the number of operations that access disks. In this project, you will practice using a B-Tree structure Specifically, after working on this project you should:

- Be able to trace insertion into and deletion from B-Trees
- Understand how to implement B-Tree functionality given a structure to work with
- Have a deep understanding of the logic behind removing from a B-Tree

2 Project Overview

You should complete this project in a group – you may select group members or be assigned a group member if you have not picked a group by the class period after this project is released. You are individually responsible for the learning goals above (dividing and conquering may not be the most effective strategy here). Use the Canvas guide on using Github to help manage version control. You may discuss this assignment with your partner(s), the course TA or instructor, but the work you submit must be your group members' own work. You may use your previous projects in this class as a reference.

2.1 B-Tree Tracing

1. In a file called `btree_insert.txt`, represent the result of inserting into a BTree with t = 2 the following values in the order listed:

   F,J,D,H,B,C,I,G,A,E,K,L,M

For each value, show the BTree at the completion of the call to `B-Tree-Insert(T,k)`.

Each level of the tree should correspond to a line. Keys within a node should be comma separated while nodes on the same level should be separated by a -. An empty line should separate each of the 13 trees. For example, a text file representing the 5 trees in figure 18.7 in CLRS would contain:

```
G,M,P,X
A,C,D,E-J,K-N,O-R,S,T,U,V-Y,Z

G,M,P,X
A,B,C,D,E-J,K-N,O-R,S,T,U,V-Y,Z

G,M,P,T,X
A,B,C,D,E-J,K-N,O-Q,R,S-U,V-Y,Z

P
G,M-T,X
A,B,C,D,E-J,K,L-N,O-Q,R,S-U,V-Y,Z

P
C,G,M-T,X
A,B-D,E,F-J,K,L-N,O-Q,R,S-U,V-Y,Z
```

2. Using the same notation as with your trees from insert, in a file called `btree_delete.txt`, represent the result of deleting from the BTree at the end of `btree_insert.txt` the following values in the order listed:

   M,I,H,B,E

2.2 B-Tree Class

I have provided the beginning of an implementation of a (non-templated) `BTree` class in `btree.h` and `btree.cpp` along with a starter test file `test_btree_example.cpp` and test case input and expected output files.

In `btree_delete.cpp`, implement the following methods for the `BTree` class:

- `remove(k)`: deletes the key k from the BTree.
- `remove(x, k, x_root)`: deletes the key k from a BTree rooted at a node x (`x_root` is a boolean that is true when x is the root of the tree).
- `find_k(x, k)`: returns the index i of the first key in a BTree node x where $k \leq x.keys[i]$. Return $i = x.n$ if no such key exists.

- `remove_leaf_key(x, i)`: removes the key at index i from a BTree leaf node x.
- `remove_internal_key(x, i, j)`: removes the key at index i and child at index j from a BTree internal node x
- `max_key(x)`: returns the maximum key in the BTree rooted at x.
- `min_key(x)`: returns the minimum key in the BTree rooted at x.
- `merge_left(x, y, k)`: merges key k and all keys and children from y into y's left sibling x.
- `merge_right(x, y, k)`: merges key k and all keys and children from y into y's right sibling x.
- `swap_left(x, y, z, i)`: gives y an extra key by moving a key from its parent x down into y, moving a key from y's left sibling z up into x, and moving the appropriate child pointer from z into y. Let i be the index of the key dividing y and z in x.
- `swap_right(x, y, z, i)`: gives y an extra key by moving a key from its parent x down into y, moving a key from y's right sibling z up into x, and moving the appropriate child pointer from z into y. Let i be the index of the key dividing y and z in x.

To address some ambiguity and ensure consistency for unit testing, please follow the logic from CLRS directly. Additionally, in cases 3a and 3b please check for an immediate right sibling first.

3 Project Submission

On Canvas, before the deadline, you will submit **a zip** containing:

- Your `btree_delete.cpp` file with your C++ implementation.
- `btree_insert.txt`
- `btree_delete.txt`

4 Grading Scheme

The out-of-class submission will be graded using the following scheme:

Criteria	**Description**	**Points**
Completeness	Meets submission requirements (files, documentation, etc.)	2
Correctness	Passes Dr. Currin's extended tests and follows specs	4
Insert	Demonstrates correct tracing	2
Delete	Demonstrates correct tracing	2
Total		**10**

The in-class component is worth 20 points. You should not need to study extra or memorize your out-of-class work for the in-class component. Working on the out-of-class component with the learning goals in mind is your best preparation. The in-class project component lets you test how prepared you are to apply the concepts covered by the project to a new scenario.