CS271: Data Structures

Instructor: Dr. Stacey Truex

Project #5

This project is meant to be completed in groups. You should work in your Unit 4 groups. Implementation solutions should be written in `C++`. Exercise solutions should be included in `.txt` files. Only one submission (the last submission uploaded to canvas) will be graded per group. Submissions should be a compressed file following the naming convention: `NAMES_cs271_project5.zip` where `NAMES` is replaced by the first initial and last name of each group member. For example, if Dr. Truex and Dr. Miller were in a group they would submit one file titled `STruexAMiller_cs271_project5.zip`. **You will lose points if you do not follow the course naming convention**. Your `.zip` file should contain a *minimum* of 4 files:

1. `btree_insert.txt`
2. `btree_delete.txt`
3. `btree_delete.cpp`
4. `commits.pdf`: a commit history for your GitHub project

Additional files such as a `README.md` are welcome. The above merely represent the minimum files required for project completion. Your code is expected to implement a `remove` function for the provided `BTree` class. Details for each part of the project are as follows.

Inserting with BTrees

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) in a file titled `btree_insert.txt`. 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 on page 510 of your textbook would appear as follows:

```
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
```

Deleting with BTrees

Represent the result of deleting from the BTree at the end of `btree_insert.txt` the following values in the order listed:

$$M, I, H, B, E$$

Use the same notation as with your trees from insert in a file titled `btree_delete.txt`.

BTree Delete Implementation

As part of the assignment, you have been provided a header file `btree.h` as well as the implementation for some helper functionality in `btree.cpp`. Implement each of the following operations in the provided `btree_delete.cpp` file to support the `remove` operation:

- `remove(k)`: deleting a key k from the BTree.
- `remove(x, k, x_root)`: deleting a key k from a BTree rooted at a node x (when `x_root` is `true` then x is the root of the tree).
- `find_k`: determine the index i of the first key in a B-Tree node x where $k \leq x.keys[i]$. Return $i = x.n$ if no such key exists.
- `remove_leaf_key(x, i)`: remove the key at index i from a B-Tree leaf node x.
- `remove_internal_key(x, i, j)`: remove the key at index i and child at index j from a B-Tree internal node x.
- `max_key(x)`: return the maximum key in the B-Tree rooted at x.
- `min_key`: return the minimum key in the B-Tree rooted at x.
- `merge_left(x, y, k)`: merge key k and all keys and children from y into y's left sibling x.
- `merge_right(x, y, k)`: merge key k and all keys and children from y into y's right sibling x.
- `swap_left(x, y, z, i)`: give y and 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)`: give y and 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 your textbooks directly. Additionally, in cases 3a and 3b please check for an immediate right sibling first.

Unit Testing

An example test file `test_btree_example.cpp` has been provided along with example test files in a `tests` folder as well as results in a `results` folder. Syntax is consistent with the example exercises you completed above.

Documentation

The expectation of all coding assignments is that they are well-documented. This means that logic is documented with line comments and method pre- and post- conditions are properly documented immediately after the method's parameter list.

Pre-conditions and post-conditions are used to specify precisely what a method does. However, a pre-condition/post-condition specification does not indicate how that method accomplishes its task (if such commenting is necessary it should be done through line level comments). Instead, pre-conditions indicate what must be true before the method is called while the post-condition indicates what will be true when the method is finished.

Rubric

Note that any coding projects that do not compile with the provided `test_btree_example.cpp` file will be given a 0. No changes should be made to `btree.h`, `btree.cpp`, or `test_btree_example.cpp`. All projects that are able to be successfully compiled will be graded using the following rubric.

<table>
<tr><td rowspan="2">Insert</td><td colspan="3">8 Total Points</td></tr>
<tr><td colspan="3">correct trees provided in <code>btree_insert.txt</code></td></tr>
<tr><td rowspan="2">Delete</td><td colspan="3">8 Total Points</td></tr>
<tr><td colspan="3">correct trees provided in <code>btree_delete.txt</code></td></tr>
<tr><td rowspan="7">C++ Implementation</td><td colspan="3">does not compile: 0/24</td></tr>
<tr><td colspan="3">24 Total Points</td></tr>
<tr><td>Code</td><td>Correctness
passes unit testing</td><td>21 pts</td></tr>
<tr><td rowspan="5">Documentation</td><td>Documentation</td><td>3 pts</td></tr>
<tr><td>extremely sparse documentation</td><td>0/3</td></tr>
<tr><td>missing comments or pre- and post-conditions</td><td>1/3</td></tr>
<tr><td>documentation lacks detail in areas</td><td>2/3</td></tr>
<tr><td>detailed comments & pre- and post-conditions</td><td>3/3</td></tr>
</table>