

1 B-Tree

1.1 Summary

You will design and implement a B-Tree Structure.

The details of the design of the B-tree structure must conform to the standard constraints of a B-Tree as discussed in class. Major factors to consider for the design should be theoretical efficiency (both time and space), practical efficiency, and appropriate memory management. Although some tips are provided below (and have been provided in class), the determination of *how* to make the structure efficient is largely your charge.

1.2 Programming Languages

I encourage you to submit your projects using C++. You may choose a different programming language with prior approval from me. However there are caveats as not all programming languages have the same characteristics. Also note, if choosing a language other than C++, you may by chance choose a language which the TAs are not familiar, thus limiting the amount of assistance they might provide. Note one of the main goals of our class projects is for you to learn how to construct various data structures from the most *elemental programming constructs*. Thus you will not receive credit when using any pre-existing structures from programming libraries or code that has been created or designed by others. For example in C++ you **cannot use** pre-existing such as vectors, stacks, lists, trees etc. To help facilitate io and timing, you may use ctime, string, ifstream, iostream, and stringstream.

Note: There are many versions of C++. You must use the version that is currently running on the course server.

Please note that complex data structures (non-elemental constructs) are “built-into” some programming languages. If this is the case, you cannot use the built-in structure. For example, in Python, both list and “array” structures are fairly complex and not elemental programming constructs, e.g., they can change size dynamically. If you are using Python, you will not receive credit when using these structures.

If you have any questions as to what structures are permitted (and which are not permitted), given your language of choice, please ask me.

1.3 Planning and Design

Before implementation, you should plan and design using standard approaches, e.g. UML class diagrams, flow diagrams, etc. If you have questions pertaining to your project, I will

first ask to see your designs. **I will not look at your code without first viewing your design documents.**

You will be faced with many design decisions during this project. It is best to spend the requisite time during the design stages to assure an appropriate and efficient implementation is built. Consider your options, perform a theoretical complexity analysis of the different options, and base your decision on the results of your analysis.

1.4 Input Requirements

The program will take one command line argument

```
./p4 [inputFile]
```

Argument 1 is inputFile: the first line of this file will contain the integer m, which determines the structure for the m-ary B-tree. The next line will contain space separated ints that are to be inserted into the B-tree, thus initializing the B-tree.

After initialization, the user should be repeatedly prompted to input a command. Valid commands are add, remove and quit. The user will type in two tokens: the first is the command, the second is the value of the key. Use the following syntax “a 6” represents command add key 6 to the B-tree; “r 10” represents command remove key 10 from the B-tree; and “q” represents command quit program. After each command, a string representation of the resulting B-tree should be printed to the screen – see details below.

1.5 Structural and Operational Requirements

1. B-Tree Implementation

- Appropriately designs a b-tree structure to allow for the creation of a b-tree of order m (for any non-negative int m).
- Must provide for basic b-tree operations: insert, remove, and display (print to console). Provide a print method or overload the “<<” operator. print a display that indicates the structure of the B-tree. I encourage you to print a “bracketed” representation of the B-Tree, which can be rendered using the the tool found at <http://mshang.ca/syntree/> . To construct such a print statement, perform a Depth First Traversal and 1) print an open bracket when traversing down 1 depth to a child node; 2) the print the priority representing that node; and 3) print a close bracket for each depth retreated.
- Appropriately allocate / deallocate memory
- Valid state of a b-tree is always maintained

Notes: To earn full marks it is expected that you will make the b-tree structure very efficient (in space and time).

1.6 Output Requirements

The following steps should be executed by the main method:

1. Initialize m-ary B-tree structure using information read from inputfile.
2. Repeatedly prompt user, add or remove items from the b-tree, and print string representation for updated B-tree.

1.7 Submission and Compilation Requirements

You must fill out a cover letter for all project submissions – See Canvas. Submission Deadline – See Canvas. Budget your time well. Include significant time for design / planning and testing / debugging. Please submit early and often! Your last submission will be graded.

Your code must run on the class server.

To standardize submissions, you will submit a makefile, which will contain the necessary compilation commands for your code. The target executable will be named p4.

Thus, the following steps should run your code on the course server (CHECK IT).

```
make p4
```

```
./p4 [inputFile]
```

If the program does not compile (following your instructions) or the program does not run, the submission will not be accepted.

1.8 Testing and Debugging (Optional)

You may wish to construct an interactive interface to test the functionality of your structure at intermediate stages of development. This would likely be most efficient with an interactive interface that allowed you to interactively test various functionalities of your structure given different inputs. *If you do implement a testing interface, please be sure to comment it (so that it does NOT execute) before submission.*

1.9 Rubric

Efficiency should be considered.

| List of Requirements | Percentage |
|---|------------|
| b-tree Design and memory allocation | 0.40 |
| Balance is maintained using appropriate reorganization scheme | 0.40 |
| Print String is correct | 0.10 |
| main method conforms to specs | 0.10 |
| TOTAL | 1.0 |