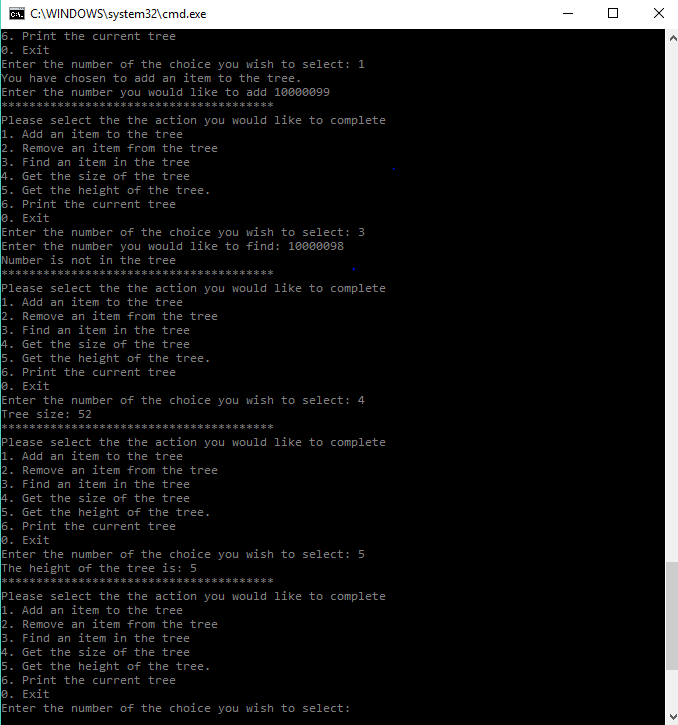
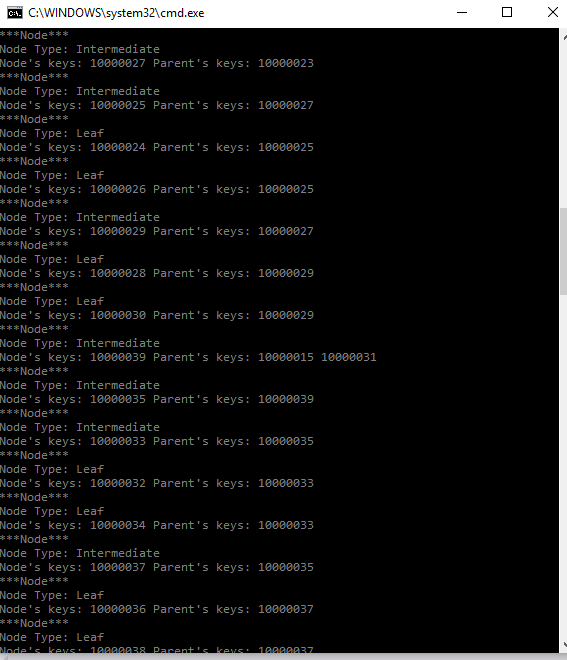
Lab10 B-Tree

Task 2 Screenshots:





Task 3:

|  |  |  |
| --- | --- | --- |
| Degree 3 | Degree 4 | Degree 5 |
| Add – 117 Find – 200 | Add – 94 Find – 150 | Add – 73 Find – 101 |
| Add – 133 Find – 200 | Add – 92 Find – 150 | Add – 77 Find – 101 |
| Add – 122 Find – 200 | Add – 105 Find – 150 | Add – 77 Find – 101 |
| Add – 135 Find – 200 | Add – 100 Find – 150 | Add – 76 Find – 101 |

The chart shows how many pointers were followed to either find the node with a key or add a key to a node. The code adds 50 keys and then finds the same 50 keys. From the table, clearly finding a key requires the code to follow more pointers than while adding. This is because when adding there aren’t as many pointers already in the tree as opposed to when searching for an item. Now as for degree 3 through degree 5, the consistency of the numbers increases as the degree increases and there will be less searches through the tree for items.

Trees alone are an important concept to computer science and are a critical data structures to know how to use. Although a software developer may never actually develop one, they might at times have to work with the data structure itself and it is important to know how it works. Compared to a binary tree, a B-tree can have multiple key values per node, and have multiple children. This speeds up the process, and minimizes the number of times the data must be accessed in order to find, add, or remove an item. A B-tree is controlled by its degree number. A tree with degree *n*, can have *n* -1 keys and *n* children. There are also other rules that control how a B-tree works because it is also a self-balancing tree. So, the data structure should always be balanced. All of these concepts are important to engineering and CS because building something or knowing that something works as efficiently as possible is important to how software is developed.

From task 2 to task 3 a few things needed to be changed in order to produce results. The first thing would be to output some sort of marker in each of the insertItme and findItem methods in the BTree template class. The next thing that needed to be done was to create two for loops that added randomly generated mNumbers and added them to the tree. Each of these mNumbers is then added to an array of mNumbers. This array is used to search the tree for those same values, with a for loop. This will generate the proper markers that will be printed.

I think the results of task 3 were what they were, because of the way BTrees are. The degree determines more-or-less the height of the tree. In that if 50 items are added to a tree of degree 3, there will be more levels than if they were added to a tree of degree 5. Having a higher degree shortens the amount of levels the program has to descend through to find/ add the value. The find counter has the same amount every time because the same amount of numbers are being added every time so the tree should have to follow the same number of pointers every time, which is shown. The higher degrees also seem to be more consistent with finding the items that are being looked for. A good question to ask would be, what degree would have highest consistency while still remaining practical in the sense of using a B-tree as a data structure.

Compilation Instructions

This has been tested by creating a new project within Visual Studios with the following options:

Win32 Console Application

Create directory for solution OFF

Empty project ON

Precompiled header OFF

SDL OFF

Then:

Add the following files to the projext:

1. BTree.h
2. OrderedList.h
3. Lab10MainTest.cpp

\*Testing (task 5) is added to both the cpp’s

Build and run

Contribution of Team Members

Everyone worked on the lab10 evenly

Evan Akers is responsible for Task1

Saylee Dharne is responsible for Task2

Smit Patel is responsible for Task3

Kyle O’Connor is responsible for the Lab Report