COP 3530

Project 2

4/13/15

Nicola Frachesen

Documentation

My code is an implementation of a bTree of strings, with a bTree being a data structure in which keys and values are inserted into a tree in which each node has a given size (number of links), and the maximum number of key/value pairs in each node is size minus one. The code I created is organized largely how it was asked that we create it, though I made the tree and node into two different classes, and added a number of helper methods. I have two files: bTree.h, containing the header for both classes, and bTree.cpp, containing the code for both classes. Here are all of the variables and methods in each class and what they do:

bTree:

Variables:

-int size: This is the maximum amount of links to child pointers that each node can have. It is input by the user.

-int minDegree: This is the minimum degree of the tree, which is calculated from the size, and is used for some of the bTree's calculations, including the minimum amount of key/value pairs in a node.

-bTreeNode *root: A pointer to the node which is currently the root of the bTree.

Methods:

-bTree(int n): The constructor for the bTree class. Takes in size, sets the minimum degree accordingly, and makes the root an empty node for now.

-~bTree(): The destructor for bTree. The only thing to delete in this class is the root.

-void insert(string key, string value): Inserts a key/value pair into the bTree. Uses the node's insert and split methods.

-bool find(string key, string *value): Searches for the given key, and has the given value pointer point to the correct value if the key is found. Returns true if the key is found, false if it is not found or if the tree is empty. Uses the node's find method by starting at the root.

-bool delete_key(string key): If the given key is in the bTree, then that key/value pair will be deleted, and true returned. If the given key is not found, then false is returned. Uses the node's delete_key method by starting at the root.

-string toStr(): Return the keys of the bTree in alphabetical order. Uses the node's traverse method to traverse the tree, starting at the root.

bTreeNode:

Variables:

-int size: This is the maximum amount of links to children pointers that each node can have. It is input by the user. Taken from the master bTree class.

-int minDegree: This is the minimum degree of the tree, which is calculated from the size, and is used for some of the bTree's calculations, including the minimum amount of key/value pairs in a node. Taken from the master bTree class.

-int keySize: The current amount of keys in this node.

-string *keys: The array of keys in this node. Corresponds to the values array.

-string *values: The array of values in this node. Corresponds to the keys array.

-bTreeNode **children: The array of pointers to the child nodes of this node.

-bool isLeaf: Keeps track of whether or not this node is a leaf. True if this is a leaf, false if this isn't a leaf.

Methods:

-bTreeNode(int n, int m, bool l): The constructor for the node class. Takes in size, minimum degree, and whether or not the node is currently a leaf. Initializes the three arrays for key, size, and child node pointers, and sets the initial keySize to be 0.

-~bTree(): The destructor for the node. Deletes the key, size, and child node pointers arrays.

-string traverse(): Returns a string of the keys in the subtree rooted at this node in alphabetical order. Recursively adds to the string by calling this method on its children. Used by the toStr method.

-bool find(string key, string *value): Searches for the given key in the subtree rooted at this node, and has the given value pointer point to the correct value if the key is found. Returns true if the key is found, false if it is not found. Used by the tree's find method.

-void insert(string key, string value): Inserts a key/value pair into the subtree rooted at this node. Used by the tree's insert method. Uses the split method if a child node needs to have something inserted into it, but is full. Also uses itself recursively to insert into child nodes.

-void split(int I, bTreeNode *x): Splits a child node pointer, x, at the current index, i, into two children. Helper method called by both insert methods if a node is full.

-bool delete_key(string key): Deletes a key/value pair in subtree rooted at this node. If the given key is in this subtree, it and its value are deleted, but if it is not there, then false is returned. Used by the tree's delete_key method. Uses the deleteNonLeaf and fill methods.

-void deleteNonLeaf(int i): Takes in an index and deletes the key/value pair at that index. This method deals with the cases which can occur when the key/value to be deleted isn't in a leaf. Uses delete_key, getPrevious, and getNext. Helper method called by the node's delete_key.

-string* getPrevious(int i): A helper method for delete_key which finds the key/value that precedes the key at the given index in order, and returns the found key/value in a string array of size 2. Helper method for deleteNonLeaf.

-string* getNext(int i): A helper method for delete_key which finds the key/value that follows the key at the given index in order, and returns the found key/value in a string array of size 2. Helper method for deleteNonLeaf.

-void fill(int i): Fills a node if it has less than minimum degree minus 1 amount of keys. Uses takeFromPrevious, takeFromNext, and merge. Helper method for delete_key.

-void takeFromPrevious(int i): Takes a key/value from the previous child (at index i-1), and places a key/value in the current child. A helper method for fill.

-void takeFromNext(int i): Takes a key/value from the next child (at index i+1), and places a key/value in the current child. A helper method for fill.

-void merge(int i): A helper method for fill that merges two children at index positions i and i +1 that are too small.

I created test cases to test and make sure each of the above methods is functional. Since I initially had some difficulty with how to have the tree work for odd numbers and size 2 (i.e. being a binary tree), I added test cases for even and odd sizes, and also one for size 2 just in case. These problems arose because I did not fully understand the concept of minimum degree, but have since been fixed. Here are my major test cases:

What Is Being Tested	Input	Expected Output	Is the Expected Output Achieved?
Basic test for insert and toStr.	-Size: 4 -Insert: Alpha A Beta B Car C Delta D -toStr	Alpha Beta Car Delta	Yes

Basic test for delete_key.	-Size: 4 -Insert: Alpha A Beta B Car C Delta D -Delete: Car -toStr	Alpha Beta Delta	Yes
Basic test for find.	-Size: 4 -Insert: Alpha A Beta B Car C Delta D -Find: Alpha Delta Beta	A D B	Yes
Large input into an even-sized bTree. Inserts enough keys to test to make sure that the insert and split methods works properly.	-Size: 4 -Insert: Go Gators Alpha A Zeta Z Beta B Car C Delta D Aardvark AA Epsilon E -toStr	Aardvark Alpha Beta Car Delta Epsilon Go Zeta	Yes
Large input into an odd-sized bTree. Inserts enough keys to test to make sure that the insert and split methods works properly.	-Size: 5 -Insert: Go Gators Alpha A Zeta Z Beta B Car C Delta D Aardvark AA Epsilon E -toStr	Aardvark Alpha Beta Car Delta Epsilon Go Zeta	Yes
Delete in an even-sized bTree. Deletes enough keys to ensure that delete_key and all of its helper methods	-Size: 6 -Insert: Go Gators Alpha A Zeta Z	Aardvark Alpha Beta Boat Car	Yes

(deleteNonLeaf, getPrevious/Next, Fill, takeFromPrevious/Next, merge) are functional.	Beta B Car C Delta D Aardvark AA Epsilon E Boat Bo -toStr -Delete: Zeta Go Alpha Boat Delta -toStr	Delta Epsilon Go Zeta Aardvark Beta Car Epsilon	
Delete in an odd-sized numbered bTree. Deletes enough keys to ensure that delete_key and all of its helper methods (deleteNonLeaf, getPrevious/Next, Fill, takeFromPrevious/Next, merge) are functional.	-Size: 5 -Insert: Go Gators Alpha A Zeta Z Beta B Car C Delta D Aardvark AA Epsilon E Boat Bo -toStr -Delete: Zeta Go Alpha Boat Delta -toStr	Aardvark Alpha Beta Boat Car Delta Epsilon Go Zeta Aardvark Beta Car Epsilon	Yes
More complex input ensuring that find works correctly.	-Size: 4 -Insert: Go Gators Alpha A Zeta Z Beta B Car C Delta D Aardvark AA Epsilon E -Find: Alpha Go	A Gators D E AA	Yes

	Delta Epsilon Aardvark		
Tests to ensure that a bTree of size 2 (essentially a binary tree) is functional.	-Size: 2 -Insert: Go Gators Alpha A Zeta Z Beta B Car C -toStr -Delete: Beta Zeta -toStr -Find: Alpha Car	Alpha Beta Car Go Zeta Alpha Car Go A C	Yes
Tests all methods to ensure that they are working well together.	-Size: 5 -Insert: Go Gators Alpha A Zeta Z Beta B Car C Delta D Aardvark AA Epsilon E Boat Bo -toStr -Delete: Zeta Go Alpha Boat Delta -toStr -Find: Epsilon Beta Car	Aardvark Alpha Beta Boat Car Delta Epsilon Go Zeta Aardvark Beta Car Epsilon E B C	Yes

Thus, all of my test cases returned as expected, and I have no known errors.