Project 3 B+ Tree Design Document.

Introduction

The program is a simple B+ Tree program. That reads in a file and determines whether what it reads in is a string or a int and if either then it will run it through the B+ tree structure and with the use of priority queue it will print them to a file. The user can also enter in strings or ints into the tree manually if they so wish. The program can also add and delete records manually. It can also search existing tree's for if the value entered is the same as a value in the tree for both strings and int types. It can also display the contents of the tree for both types of trees.

Data Structure

The node data structure is utilized by other data structures to organize data. Each node has an array records consisting of a key and references to the node below it. The nodes also have references to the parent, first and next nodes. The nodes can split to produce more nodes. Additionally the nodes can split their parent nodes. The data in each node can be output to the terminal.

The B+ tree is the primary data structure in this program. The B+ tree is composed of nodes.The b+ tree unlike many trees is built from the leaves up. When a new data enters the B+ tree, starting with the root, the program navigates down through nodes to find the proper place for the data. If there is room for the data in that node then the data are inserted and the node above denotes change to the node if need be. However if there is no room left in the node the node will split into two nodes and create a new node above those nodes.When data would be removed from a node that would cause it to be smaller than the half the max size the left and right nodes merge.The B+ tree structure can also be output to the terminal. Alternatively the structure can output all of the keys to the terminal.

A queue data structure is used to help organize data from the B+ tree so the can be displayed. The queue holds data in a first in first out structure. Data is enqueued into nodes. References to the nodes are held in an array. The structure tracks the head and tail of the queue and moves them when data is enqueued and dequeued.

Functions:

Functions in Main.cpp:

Void menu(): is a function that displays a menu for different options for different operations on the tree

Void helpMenu(): is a function that displays helpful commands if the user needs them.

Functions in shared.h:

Std::ofstream mount(“out.txt”): is a function that creates the output file that serve as an output stream to write to an output log file.

Functions in node.h:

node(): is a Default node constructor for the Class node

Void nodeInsert(record value): is a function that inserts a record in a node that has a key that that is of type int.

Void stringNodeInsert(record value): is a function that inserts a record in a node that has a key that is of type string.

record splitNode(record value): is a function that splits a node when there is no more room to add a record. This is for only keys that are of type int.

record stringSplitNode(record value): is a function that splits a node when there is no more room to add a record. This is only for keys that of type string.

record splitParent(record value): is a function that splits the parent. This is only for keys that are of type int.

record stringSplitParent(record value): is a function that splits the parent. This is only for keys that are of type string.

Node *nextindex(int value): is a function that takes the next key of the value of type int.

Node*stringNextindex(string value): is a function that takes the next key of the value of type string.

Void display(): is a function that displays the tree structure visually. This only displays the tree structure for int.

Void stringDisplay(): is a function that displays the tree structure visually. This only displays the tree structure for strings.

Void displayval(): is a function that displays that values of the keys. This will only display the value of keys of type int.

Void stringDisplayval(): is a function that displays the values of the keys. This will only display the values for keys of type string.

Functions in Queue.h:

Queue(): is a Default Constructor function

Int empty(): Is a function that checks whether the queue is empty or not.

Node *deque(): is a function that dequeues to the queue.

Void enque(node *value): Is a function that enqueues to the Queue.

Void makeEmpty(): is a function that makes the queue empty.

Functions in bPTree.h:

bPTree(int n):Is a default constructor of the Class bPTree.

Void insert(int value): Is a function that Inserts a key of type int into the tree.

Void stringInsert(string value): is a function that inserts a key of type string into the tree.

Void showTree(): is a function that displays the tree with keys of type int.

Void stringShowTree(): is a function that displays the tree with keys of type string.

Void listValues(): is a function that goes through the tree to display all of the key values of type int.

Void stringLIstValues(): Is a function that displays all of the key values of type string.

Void intdelete(int value): is a function that removes a key of type int from the tree.

bool search(int value): a Function that searches the tree of type int and finds the value of type int.

The Main Program

The main program initializes the size of the block, a key value of type int and a key value of type string. The main program also performs the arg Execution data. For example if (argc ==3) and if argv[1][1] == ‘s’) which is the input file if it is of type string, Then it will then it will initialize the ifstream stringInfile(argv[2]); and print out and mount the message String file opened successfully. It will then print out a prompt for the user to enter the number of values per block which it will also mount. It will then print out and mount a message saying the number of values per block have been set to what the user typed in and then it will call the class function bPlusTree bPlusTree with the number the under entered being used within it. It will then print and mount the message telling the user the Loading keys into the tree. Then while the value you being read in the input file is a string value then it will call the function bPlusTree.stringInsert(stringValue); when it finished reading in the values it will then print out and mount the message The B+ tree has been created. After that it will initialize the values int choice = ‘0’ and string value. Then it will run a while statement that will run while choice is less than 5. During this it will call the function menu() and prompt the user to enter in the number for choice. Then it will run a switch statement called switch(choice) where is case is ‘1’ it will then go through the process of adding a new record. If it’s case ‘2’ then it will go through the process of deleting a record. If it’s case ‘3’ it will then show the tree and if it’s case ‘4’ it will list the values within the tree. If it’s case ‘5’ it will exit the program. If it’s the default case it will print out and mount the message “something went wrong..” and then it will exit the program. If(argv[1][1] = ‘i’) which will run if input file is of type int then it will do the exact same as the process up above for type string expect that value is of type int. After both of the if statements close. Then If (argc == 2) which is for a empty tree. Then it will run another one of two if statements if(argv[1][1]= ‘s’) which will run if empty tree is of type string or if(arg[1][1] == ‘s’); if the empty tree is of type int. It will then proceed to go through the same process as what was stated before for either of the if statements. When that if statement close’s or get’s passed then if(argv[1][1] == ‘h’); then it call the function helpMenu().