Aberystwyth University/Northeastern University

Data Structures

Assignment: Implement 2-3 Trees using Arrays

This assignment will contribute 50% to the assessment for the
Data Structures course

Introduction

In some application areas, pointers are not allowed to be used because of potential problems of memory management. For example, pointers are often not allowed to be used in safety critical applications. In such application areas, all data structures have to be built using arrays.

This assignment is concerned with implementing the 2-3 Tree abstract data type. For simplicity we will assume that the tree is to hold integer values.

Each tree node must contain: a pointer to its parent (if any), an integer representing the number of keys in the node (which is always one or two for a properly constructed 2-3 Tree), pointers to the head and tail of the list of keys, and pointers to the head and tail of the list of pointers to the child nodes (if any).

Since we are using arrays, all pointers are represented by indices into the appropriate array. We use –1 to represent a null pointer.

The list of keys and the list of pointers to the child nodes are doubly-linked lists (DLLs), that is, they have forward and back pointers.

So, a list node could be pictured as shown here: where previous and next are of type int.

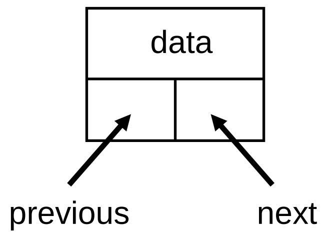


Figure 1

A DLL itself is represented by an array (of suitable length) of nodes, and values head and tail of type int, indicating the position of the first and last elements of the list in the array (respectively).

For example, after inserting 23, 32, 17 into the list (in this order), starting at index position 0, and maintaining the list in ascending order, the array will look like:

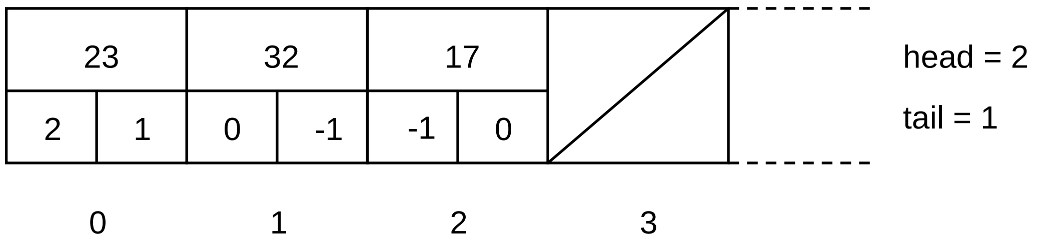


Figure 2

and the list will look like:

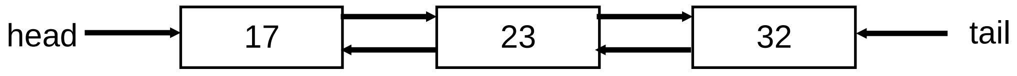


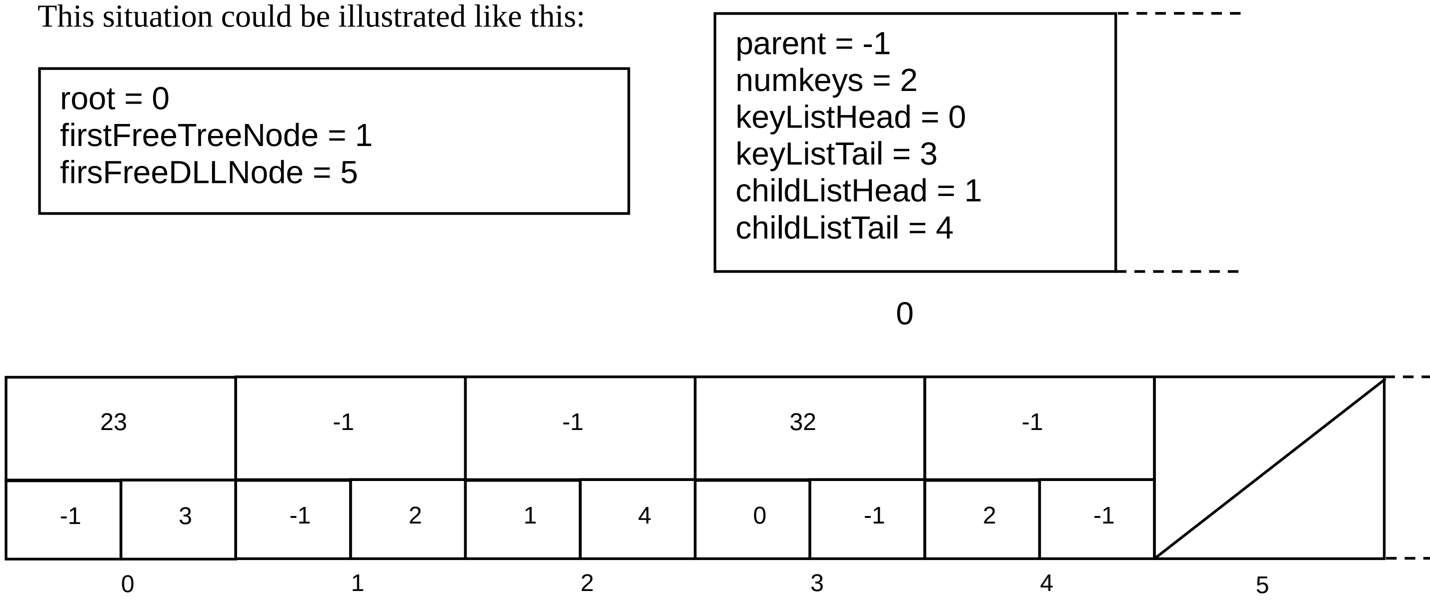
Figure 3

The 2-3 Tree itself consists of an array of tree nodes, an array of DLL nodes, and values root, firstFreeTreeNode, and firstFreeDLLNode, all of type int, representing:

- the index in the array of tree nodes of the root of the tree;
- the location in the array of tree nodes of the first free location in that array, and;
- the location in the array of DLL nodes of the first free location in that array.

For example, suppose we have a 2-3 Tree with just one node containing the keys 23 and 32 (which arrived in that order). Then we might have root = 0, firstFreeTreeNode = 1, and firstFreeDLLNode = 5. In the tree node at index position 0 we might have the values parent = –1, numKeys = 2, keyListHead = 0, keyListTail = 3, childListHead = 1, childListTail = 4.

This situation could be illustrated like this:



To carry out your assignment, you should complete the following tasks:

Task 1: Implement the DLLNode Classes [20%]

Implement the following classes:

DLLNode
– data : int – previous : int – next : int
+ DLLNode (data : int) + DLLNode (data : int, previous : int, next : int) + setData (data : int) + getData ( ) : int + setPrevious (previous : int) + getPrevious ( ) : int + setNext (next : int) + getNext ( ) : int

ListOverflowException extends RuntimeException
+ ListOverflowException ( ) + ListOverflowException (message : String)

Write a test program that enables you to build a DLL of integers using an array (as described above). Your test program should include a method insertInOrder(int data) where data is the value to be inserted into the DLL. This method should add into the DLL a new node from the next free position in the array of DLL nodes and adjust various pointers (indices) so that the list is maintained in ascending order, as shown in Figures 2 and 3 above. This method will require access to, and must be able to modify, the variables head, tail, and firstFreeDLLNode.

So that you can check that your insertInOrder() method is working properly, implement a method toString() so that you can display the data, previous and next values in every occupied element of the array, in array index order. Use this toString() method in your test program to display your DLL after you have inserted a few elements so that you can check that your insertInOrder() method is working properly.

You may include additional properties and operations to those suggested above if you think that any are helpful, but you should not make your system too complicated.

Task 2: Implement the Tree Classes [40%]

Implement the following classes:

TreeNode
– parent : int – numKeys : int – keyListHead : int – keyListTail : int – childListHead : int – childListTail : int
+ TreeNode ( ) + TreeNode (data : int, leftChild : int, rightChild : int) + setParent (parent : int) + getParent ( ) : int + setNumKeys (numKeys : int) + getNumKeys ( ) : int + setKeyListHead (keyListHead : int) + getKeyListHead ( ) : int + setKeyListTail (keyListTail : int) + getKeyListTail ( ) : int + setChildListHead (childListHead : int) + getChildListHead ( ) : int + setChildListTail (childListTail : int) + getChildListTail ( ) : int

The TreeNode constructor with three parameters should construct a new tree node containing a single key with the data value given as the first parameter, and pointers to its left and right children given by the two remaining parameters.

TreeOverflowException extends RuntimeException
+ TreeOverflowException ( ) + TreeOverflowException (message : String)

The 2-3 Tree itself is represented by the following class:

TwoThreeTree
– DEFAULT_MAXIMUM : int – TreeNodeArray : [ ] TreeNode – DLLNodeArray : [ ] DLLNode – root : int – firstFreeTreeNode : int – firstFreeDLLNode : int
+ TwoThreeTree ( ) + TwoThreeTree (size : int) + insert (data : int) + toString ( ) : String

The size parameter in the second TwoThreeTree constructor specifies the length of the TreeNodeArray array. If that constructor is not used, the array should have the length specified by the DEFAULT_MAXIMUM value. The DDLNodeArray should be five times as long as the TreeNodeArray array.

The insert() method must allow a new key value to be inserted into the 2-3 Tree, and the tree adjusted so that it is still a valid 2-3 Tree. The algorithm for inserting into a B-Tree (and hence, into a 2-3 Tree) is outlined on slide 12 of the set of slides on B-Trees. There are a number of cases to consider. I strongly suggest that you write auxiliary methods to implement the various cases which are called by your insert() method. In this way, the insert() method will not become too large.

The toString() method should enable you to display all the information in a 2-3 Tree, including the used parts of the TreeNodeArray and DLLNodeArray arrays, so that you can see that everything is working properly.

Task 3: Main Program [10%]

Implement a main program that will allow you to create a 2-3 Tree, add elements to it, and run toString() and display the results. Your main program should prompt for elements to be added to the 2-3 Tree, and allow toString() to be run on request.

Run your main program and add into the 2-3 Tree the following values in this order: 23, 32, 17, 15, 39, 9, 16, 29, 35, 34, 27, 31, 33, 6, 10, 8, 7.

Run toString() on the resulting 2-3 Tree and include a screen shot of the output obtained in your assignment submission.

Run two other tests of your program using data of your own choice, including one test that attempts to insert more data than can be accommodated in the arrays used to hold your 2-3 Tree and, again, include screen shots of the output obtained in your assignment submission.

Task 4: Add an inOrder() method [15%]

Only after completing Task 3, add an inOrder() method to your 2-3 Tree class that should return all the elements in the 2-3 Tree in ascending order. This method should be a generalisation of the inOrder() method for a binary search tree.

Modify your main program to allow the inOrder() method to be called. Include in your assignment submission a screen shot of the result of calling the inOrder() method on the 2-3 Tree constructed from the data listed in Task 3.

Task 5: Compactification [15%]

Only after completing Task 3, consider the following.

While rearranging the 2-3 Tree when an element is inserted, elements of the TreeNodeArray and DLLNodeArray arrays may be discarded, leaving "holes" in the arrays. Hence, the arrays could become full even though there is space in the middle of them.

Consider ways of overcoming this problem and write a short report (no more than one side of A4 paper) on possible solutions to the problem. Include a paragraph on what impact the solutions you discuss would have on the time performance of the insert() method.

Note. This Task does not involve any implementation. You are required to only think about the problem stated above and write about possible solutions and their time performance.

Submission details

The assignment is due to be submitted to BB Blackboard as a **ZIP** file by 8:00pm on **Sunday, 10th April**. You **must** submit the following:

- Fully documented code listings.
- The screen shots from tasks 1, 3, and 4 (if completed).
- Your report from task 5 (if completed).

Please submit only a **single ZIP** file. **Please do not** attempt to submit more than one file to BB Blackboard. Also, **please do not** submit any other form of archive file, i.e., **do not** submit a **tar**, **rar**, **war**, etc., file.

Some marks will be awarded for the quality of the code submitted. It should be readable, so that I can understand what it is doing. That is, sensible variable and method names should be used, the code should contain useful comments, and it should be formatted so that it does not contain long lines that make it difficult to follow.

Time Management

It is envisaged that this assignment will take about 20 hours to complete.