Hacettepe University

Computer Engineering

BBM 203 – HOMEWORK 4
REPORT

Name: Zekeriya Onur Yakışkan
Student id: 21527539

Problem

Problem is constructing a tree structure from given inputs. Then doing operations on the tree. Constructing the tree is fundamental and must be done first. Then operations list and delete can be done with respect to constructed tree.

Constructing Tree

Tree is constructed with respect to an input file. The input file has two columns and its number of rows is not specified. Each column holds integers. First columns integers is unique with respect to other integers in the first column. First column represents data in the nodes of the tree. First rows first column is the root of the tree. Integers in the second column represent how many rows must be read from first column. If we say that the number in the second column is a(n), then we must look to first column for a(n) times and place each number starting from most left leaf to most right leaf. If program reaches the most right node and places it, it m must return to its starting leaf and continue to process.

Operations

There are two operations, delete and list. Each operation needs an integer value to perform. Operations is taken from another input file. The input file has two columns and has unspecified number of rows. First column can take two char, d for delete and l for list. Integer that this operation needs is given in the second column.

List Operation

List operation takes an integer input. Since every node has data as a unique integer, input represents a node. List operation visits the constructed tree from the given node in preorder and prints its every visit to output.txt.

Delete Operation

Delete operation takes an integer input. Since every node has data as a unique integer, input represents a node. Delete operations deletes the given node and adds its childs to the given nodes parents child. Its most left child takes the place of the deleted nodes place in parents child. If given node is a leaf then just deletes it. If given node is the root, then roots most left child becomes the root and its childs takes the place of the most left child.

Data Structures

There is two data structures in this program. First one is for holding each node of the tree. Its name is treeNode. It has three attributes which are data, child and sibling. Data stores the value of the node as an integer. Child stores the most left child of the given node as a treeNode pointer. Sibling stores the first sibling of the given node as a treeNode pointer.

```
typedef struct treeNode{

        int data;
        struct treeNode *child;
        struct treeNode *sibling;

}treeNode;
```

My second structure is named linkedNode. It can store nodes in a linked list. It is used for finding leaves.

```
typedef struct linkedNode{

        struct treeNode *cur;
        struct linkedNode *next;

}linkedNode;
```

Functions

int lenStr(char *str);//returns length of a given string
int rowNumber(FILE *ptr);//returns how many rows that a file have with substracting rows that only has a newline character.
int getFirstDigit(FILE *ptr);//applicable only for this hw
int getSecondDigit(FILE *ptr);//applicable only for this hw
void addChild(treeNode *parent, int x);
treeNode* initTreeNode(int x);//returns a treeNode which has the value of x and both child and sibling is NULL
linkedNode* initLinkedNode();//returns a linkedNode which both cur and next is NULL
int findLeaves(linkedNode *head, treeNode *root);//returns the number of leaves for testing purposes
void deleteLinked(linkedNode *head);//deletes and frees linked node
void printTree(treeNode *root, char *str);//writes the tree to a string from given node pointer, with preorder
void printLinked(linkedNode *head);//Testing Purposes
treeNode* createTree(FILE *ptr1, FILE *ptr2);//creates tree and returns its root
treeNode* findParent(treeNode *parent, int x);//finds the parent of x from given root. Returns NULL if x is not in the tree
void deleteRoot(treeNode **root);//deletes root
void deleteMidTreeNode(treeNode *root, int x);//deletes a node which is not the root. Does nothing if x is not in the tree
void deleteTreeNode(treeNode **root, int x);//deletes tree node whichs data is x. If x is not in the list, does nothing
treeNode* findTreeNode(treeNode *root, int x);//find tree node whichs data is x. If x is not in the tree returns NULL
void list(treeNode *root, int x, FILE *ptr);//prints every nodes data starting from x, with preorder
void operate(treeNode **root, FILE *ptr_in, FILE *ptr_out);//does operations delete and list with respect to given file

Important Functions

```
findLeaves(linkedNode *head, treeNode *root){
	int n = 0;
	if(root->child == NULL){
		if(head->cur == NULL)
			head->cur = root;
		else{
			for(;head->next;head = head→next);//find the last linkedNode
			linkedNode *new = initLinkedNode();
			new->cur = root;
			head->next= new;
		}
		return 1;
	}

	root = root->child;
	for(;root;root = root->sibling){

		n = n + findLeaves(head, root);
	 }

	return n;
}
```

*It visits each node exactly once (n times) and visits leaves for l(l+1)/2 times, therefore its time complexity is O(n * l^2)

```
createTree(FILE *ptr1, FILE *ptr2){
	int rowNum = rowNumber(ptr1);
	int sum = 0,first, second,i;
	linkedNode *leaves = initLinkedNode(), *mostLeftLeaf;
	rowNum--;
	treeNode* root = initTreeNode(getFirstDigit(ptr1));
	while(sum < rowNum){
		deleteLinked(leaves);
		leaves = initLinkedNode();
		findLeaves(leaves, root);
		mostLeftLeaf = leaves;
		second = getSecondDigit(ptr2);
		sum += second;
		if(sum > rowNum){
			second = rowNum - sum + second;
		}
```

```
            for(i = 0; i < second; i++){
                first = getFirstDigit(ptr1);
                addChild(leaves->cur, first);
                if(leaves->next == NULL)
                    leaves = mostLeftLeaf;
                else
                    leaves = leaves->next;
            }

        }
        return root;
}
```

*It visits every node once so its time complexity is O(n).

```
printTree(treeNode *root, char *str){

        int len = 0;
        if (root == NULL){
            return;
        }
        sprintf(str,"%d,",root->data);
        root = root->child;
        for(;root; root = root->sibling){
            len = lenStr(str);
            printTree(root, &str[len]);
        }

        return;

}
```

*Visits every child once therefore O(n)

```
deleteMidTreeNode(treeNode *root, int x){

	treeNode *parent = findParent(root , x);
	if(parent == NULL)
		return;
	treeNode *child,*grandchild,*sibling;
	if(parent->child->data == x){
		child = parent->child;
		if(child->child == NULL){
			parent->child = child->sibling;
			free(child);
			return;
		}
		grandchild = child->child;
		for(;grandchild->sibling; grandchild = grandchild->sibling);
		grandchild->sibling = child->sibling;
		grandchild = child->child;
		parent->child = grandchild;
		free(child);
		return;
	}
	child = parent->child;
	sibling = child->sibling;
	while(sibling->data != x){
		child = child->sibling;
		sibling = sibling->sibling;
	}
	if(sibling->child == NULL){
		child->sibling = sibling->sibling;
		free(sibling);
		return;

	}
	grandchild = sibling->child;
	for(;grandchild->sibling; grandchild = grandchild->sibling);
	grandchild->sibling = sibling->sibling;
	grandchild = sibling->child;
	child->sibling = grandchild;
	free(sibling);
	return;

}
 *time complexity is O(n)
```