

Binary Tree ADT

CS2263 – Systems Software Development

Everybody Hurts, R.E.M. (Automatic for the People, 1992) https://www.youtube.com/watch?v=5rOiW_xY-kc

1

Learning Outcomes

At the conclusion of this lecture students should be able to:

- List the abstract required functionality of a tree
- List and evaluate two ways of implementing a tree as an ADT
- Create a binary tree as a typedef/struct in C
- Write the CRUD functions to use a binary tree data structure using links.

2

References

- Lu, Yung-Hsiang. 2015. Intermediate C Programming. CRC Press. New York. (Chapter 20)

3

What is a Tree?

Think about a linked-list

- Parent-child relationship
- Only one child per parent
- What if there were more children?

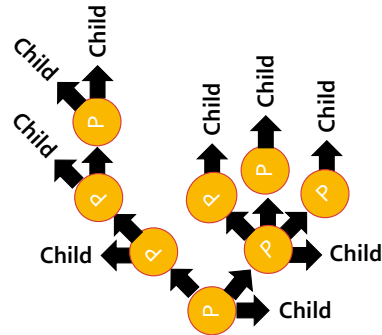


4

What is a Tree?

Think about a family tree

- Parents and children
- N -children per parent

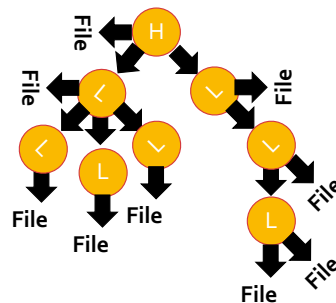


5

What is a Tree?

Think about a your home directory

- Directories and files
- N -children per parent



6

Binary Trees: data structure

A collection of nodes/links

- Maximum two children per parent

```
typedef struct btn BTreeNode, *pBTreeNode;
```

```
struct node {
    Payload* payload;
    pBTreeNode left;
    pBTreeNode right;
};
```

7

Binary Trees: functionality

- What functionality is required?
 - Create the list
 - Read ("get") items in the list
 - Update
 - Add items
 - Remove items
 - Delete the list
- Does order matter?
 - Implicitly, yes

8

Binary Tree:
Implementation
Choices

What data structures can we use to implement a binary tree?

- Array
- File
- Linked-list

Evaluation for each choice (implications for CRUD):

- Array
- File
- Linked-list

9

Why Links?

Agility

- Links/nodes
- Modified at runtime Arrays
 - Overhead?
 - Complexity
- Defined at runtime Arrays
- Compile-time Arrays

10

Binary Trees in Java

- What do you remember about Binary Trees from CS1083?
 - What are the parts?
 - How do they work?

11

Operations on Binary Tree

- **C**reate
 - A pointer to the tree (head)
 - or the creation of a Tree typedef/struct to hold it (ADT)
- **R**ead
 - Process the list
- **U**pdate
 - Add items
 - Remove items
- **D**elete
 - Remove the items
 - Remove the Tree

12

Create a Binary Tree (not a node)

As a linked-list (no Binary Tree structure)

```
Node* pRoot = (Node* NULL);
```

As an ADT (with Binary Tree struct)

```
BTree* pT = mallocBTree();
```

```
-----
BTree* mallocBTree() {
    allocate the structure
    set the head to NULL
    ? set nLinks to zero
}
```

13

Processing a Tree

The easiest way to traverse the tree is using recursion

- Recursive data structure
- Recursive functions

14

Traversal Patterns

The tree is implicitly ordered by the payload

- Payloads are distinct

Traversal Techniques

- Breadth-first
 - Uses a queue!
- Depth-first
 - In-order: left, root, right
 - Pre-order: root, left, right
 - Post-order: left, right, root

15

"Read"(access) Example

Print out the elements of a tree, in order

```
static void printfBTN(Payload *p)
{
    /*
     * print the payload, e.g.
     * printf("%d ",*p); if payload is an int
     */
}

static void printfBTNIInorder(pBTTreeNode pBTN){
    if (pBTN == (pBTTreeNode)NULL ) return;

    printfBTNIInorder(pBTN -> left);
    printfBTN(pBTN->value);
    printfBTNIInorder(pBTN -> right);
}
```

16

Insert Example

Insert a node

```
pBTNode insertBTNode(pBTNode pBTN, Payload* p)
{
    // empty, create a node
    if (pBTN == NULL) return createBTN(p);

    // not empty
    // do not insert the same value
    // Careful - this version compares addresses!
    if (p == (pBTN->payload) ) return pBTN;

    if (p < (pBTN->payload)) pBTN->left = insertBTNode(pBTN -> left, p);
    else pBTN -> right = insertBTNode(pBTN -> right, p);

    return pBTN;
}
```

17

Delete Example

Delete a node (too long for actual code)

1. No node: return NULL
2. Traverse the tree for the payload. Not found: return NULL
 1. Node has no children
 1. free()
 2. Node has both children
 1. Decide which node (left or right) to promote
 2. Promote payload, remove node (Just like linked-list)
3. Node has right-child
 1. Just like a linked-list
4. Node has left-child
 1. Just like a linked-list

18