Yasmine Siala

Professor Veenstra

CSE13S

11 November 2024

Assignment 4 Draft

tree_print_node(Node *node)

1. Purpose: To recursively print a subtree starting from a specified Node in a binary tree using traversal
2. Parameters: Node *node
3. Return value: void
4. Pseudocode:

```
    - If node == NULL, return
    - tree_print_node(node->left)
    - Print the count and key of node
    - tree_print_node(node->right)
```

tree_print(Tree *tree)

1. Purpose: Initiate the printing of a binary tree by calling tree_print_node() with the tree's root node. If the tree is empty, no output should be printed.
2. Parameters: Tree *tree
3. Return value: void
4. Pseudocode:

```
    - Check if tree or tree->root is NULL
    - If tree->root == NULL, return
```

```
- Otherwise, call tree_print_node(tree->root)
```

Tree *tree_alloc(void)

1. Purpose: To allocate memory for a new Tree structure and return a pointer to it.
2. Parameters: void
3. Return value: Tree *
4. Pseudocode:

```
- Use calloc() to allocate memory for a tree structure
- Use assert() to check that the memory allocation worked
- Initialize the root of the new tree to NULL
- Return the pointer
```

tree_add(Tree *tree, int key)

1. Purpose: Add a given key to a binary search tree. If a nose with the key already exists in the tree, the function increments the count of that node. If no node with the key is found, the function creates a new node with key, sets its count to 1 and inserts it into the correct position
2. Parameters: Tree *tree, int key
3. Return value: void
4. Pseudocode:

```
- Start with a pointer to the root node (pointer to pointer)
- While current node != NULL:
    - If the current node's key == given key:
        - Increment the count of this node by 1
        - Return
    - If the given key < current node’s key:
        - Move to the left child by updating the pointer to pointer
    - Else if the given key > current nodes key:
        - Move to the right child by updating the pointer to pointer
- If key not found in the tree:
    - Allocate a new node
    - Set the new node’s key to the given key
    - Set the new node’s count to 1
    - Set the NULL pointer to point to this new node
```

tree_free_node(Node *node)

1. Purpose: To recursively free all nodes in a subtree, starting from a given node
2. Parameters: Node *node
3. Return value: void
4. Pseudocode:

```
    - If the node == NULL, return
    - Recursively call tree_free_node() on the left child of the node
    - Recursively call tree_free_node() on the right child of the node
    - Free the current node
```

tree_free(Tree **p)

1. Purpose: To free the entire Tree structure, including all its nodes, and set the Tree pointer to NULL after freeing
2. Parameters: Tree **p
3. Return value: void

4. Pseudocode:

```
- If *p == NULL, return
- Call tree_free_node() with the root of the tree to free all nodes in the tree
- Free the memory of the tree structure itself via free(*p)
- Set *p = NULL to indicate the tree has been freed
```

check_number()

1. Purpose: verify that all characters in the given string are digits
2. Parameters: const char *s
3. Return value: void
4. Pseudocode:

```
- Loop through each character in string s:
    - If the character is not a digit, call print_usage() and return
```

main()

1. Purpose: Parse command-line arguments, add numbers to binary tree, and either print the tree or dump the tree's structure based on the command-line operations
2. Parameters: int argc, char **argv
3. Return value: int
4. Pseudocode:

```
- Allocate a new tree
- Loop over the arguments from 1 to argc - 1:
    - If the argument is "-d", set the flag to true and skip this arg
    - Else: check if the argument is a valid number using check_number()
    - Convert the argument to an integer using atoi()
```

    - Add the number to the tree via tree_add()
- After processing all arguments, if the “-d” flag was set, call tree_dump()
    - Else: call tree_print()
- Free the tree’s allocated memory