COURSE NAME: Data Structures and Algorithms

STUDENT: Ertuğrul ŞENTÜRK

HOMEWORK SUBJECT: Word Changer Application

Function Definitions

1- The address and value information of prev, next, left, right nodes are stored in the linked list structure. The prev node was used for insertion sort.

2- create_list function has been created to create a new linked list element from the given letter value.

3- add_char_to list function is the function that adds the given letter as a new element to the linked list if the given letter is not in the list, and increases the frequency of the element if it exists in the list.

4- insertion_sort function is the function that sorts the insertion list according to the frequency value.

5- convert_huffman is a function that takes the first two elements of the given list as left and right and creates a root element that takes the frequency value as the sum of these elements, and places this element in the appropriate place so that it does not disrupt the order of the linked list and repeats this process until the list is empty.

6- tree_height is a function that calculates recursively the height of a given binary tree.

7- print_level prints given level of the tree.

8- print_huffman prints each level of the tree.

Algorithm:

1- Input text is get from the user.

2- By calling the add_char_to_list function for each character of the text received, and linked list structure is created. If new element in the list that elements frequency increased, if it isn't a new node created for that character and added to the end of the list.

3- This list sorted with insertion sort by using prev and next nodes.

4- For this operation, a new root node has been created each iteration with the first two elements of the list being left and right nodes respectively.

5- The frequency value of the created root node was set to be the sum of the frequencies of these two nodes.

6- This function iterated until the last 2 items of the linked list remain.

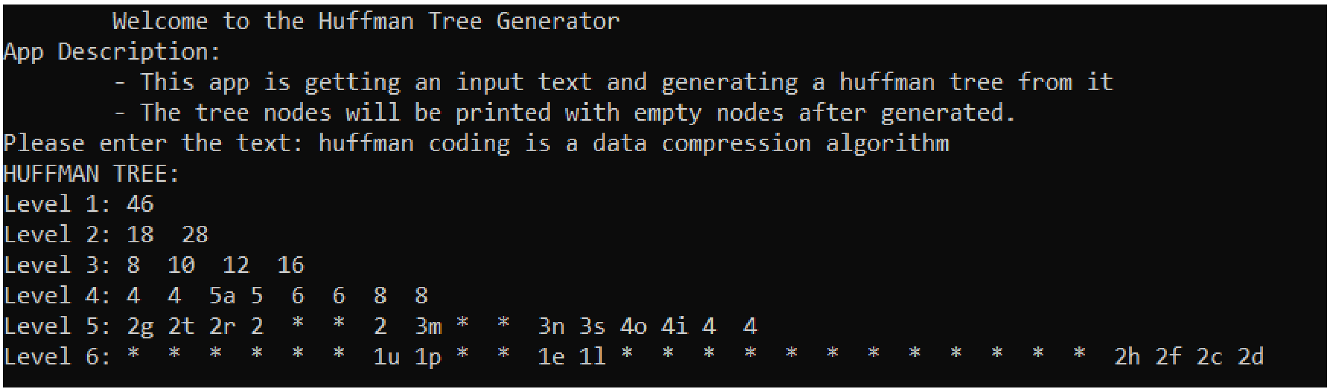
7- for the last 2 elements, the main node was created with the smaller element bounded to left and other one to the right node. In this way, the linked list structure converted into a binary tree.

8- with print huffman function tree is printed for each level.

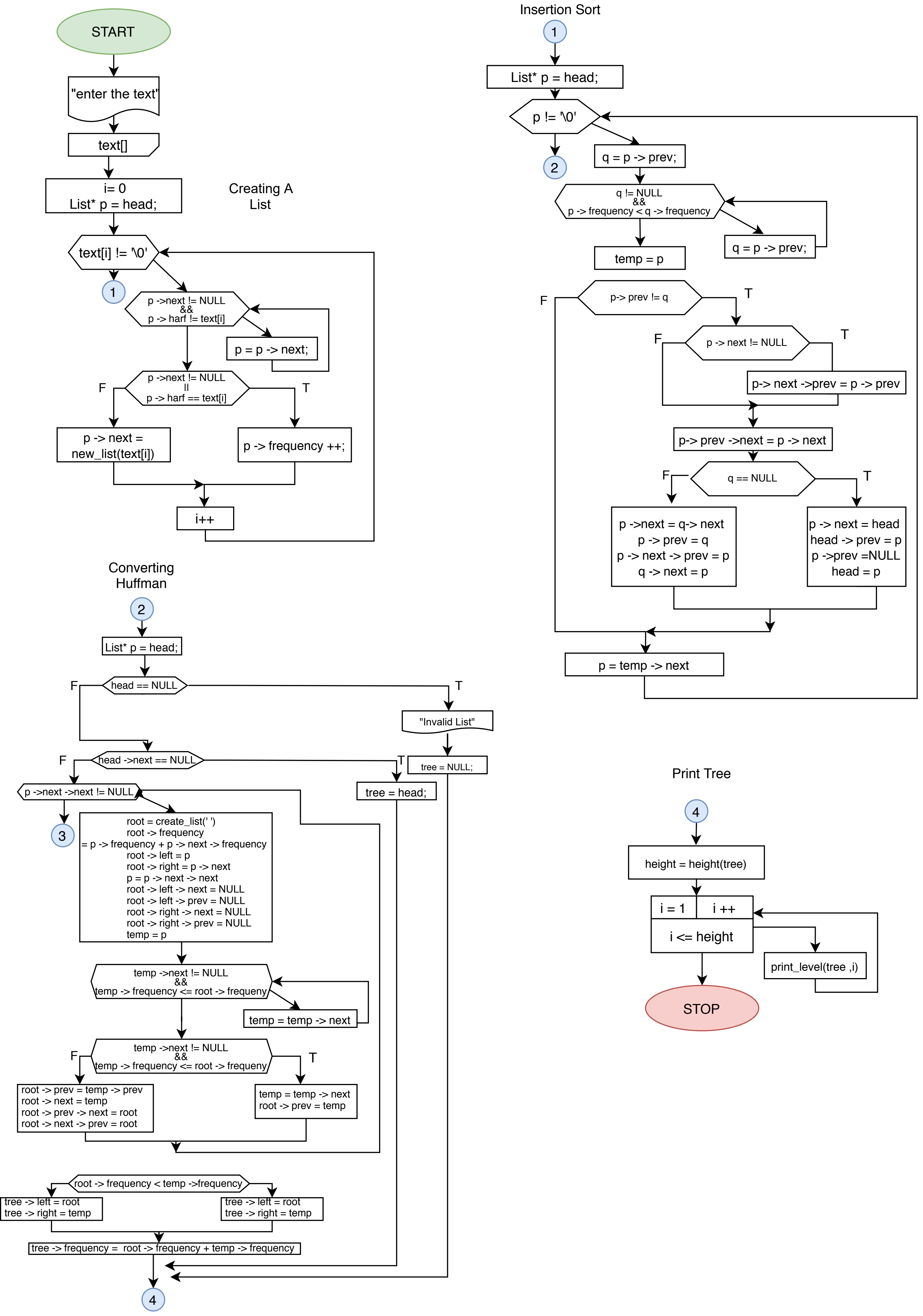
Screenshot:

Input = huffman coding is a data compression algorithm

Output:



```
        Welcome to the Huffman Tree Generator
App Description:
        - This app is getting an input text and generating a huffman tree from it
        - The tree nodes will be printed with empty nodes after generated.
Please enter the text: huffman coding is a data compression algorithm
HUFFMAN TREE:
Level 1: 46
Level 2: 18  28
Level 3: 8  10  12  16
Level 4: 4  4  5a 5  6  6  8  8
Level 5: 2g 2t 2r 2  *  *  2  3m *  *  3n 3s 4o 4i 4  4
Level 6: *  *  *  *  *  *  1u 1p *  *  1e 1l *  *  *  *  *  *  *  *  *  *  *  *  2h 2f 2c 2d
```


START
"enter the text"
text[]
i= 0
List* p = head;
Creating A List
text[i] != '\0'
1
p ->next != NULL
&&
p -> harf != text[i]
p = p -> next;
p ->next != NULL
||
p -> harf == text[i]
F
T
p -> next = new_list(text[i])
p -> frequency ++;
i++
Insertion Sort
1
List* p = head;
p != '\0'
2
q = p -> prev;
q != NULL
&&
p -> frequency < q -> frequency
q = p -> prev;
temp = p
F
p-> prev != q
T
F
p -> next != NULL
T
p-> next ->prev = p -> prev
p-> prev ->next = p -> next
F
q == NULL
T
p ->next = q-> next
p -> prev = q
p -> next -> prev = p
q -> next = p
p -> next = head
head -> prev = p
p ->prev =NULL
head = p
p = temp -> next
Converting Huffman
2
List* p = head;
F
head == NULL
T
"Invalid List"
tree = NULL;
F
head ->next == NULL
T
tree = head;
p ->next ->next != NULL
3
root = create_list(' ')
root -> frequency
= p -> frequency + p -> next -> frequency
root -> left = p
root -> right = p -> next
p = p -> next -> next
root -> left -> next = NULL
root -> left -> prev = NULL
root -> right -> next = NULL
root -> right -> prev = NULL
temp = p
temp ->next != NULL
&&
temp -> frequency <= root -> frequeny
temp = temp -> next
F
temp ->next != NULL
&&
temp -> frequency <= root -> frequeny
T
root -> prev = temp -> prev
root -> next = temp
root -> prev -> next = root
root -> next -> prev = root
temp = temp -> next
root -> prev = temp
root -> frequency < temp ->frequency
tree -> left = root
tree -> right = temp
tree -> left = root
tree -> right = temp
tree -> frequency = root -> frequency + temp -> frequency
4
Print Tree
4
height = height(tree)
i = 1
i ++
i <= height
print_level(tree ,i)
STOP