DERSİN ADI: Veri Yapıları ve Algoritmalar

DERSİN EĞİTMENİ: Doç. Dr. M. Elif KARSLIGİL

ÖDEV NO: 2

ÖĞRENCİ ADI: Ertuğrul ŞENTÜRK

ÖĞRENCİ NO: 18011028

ÖĞRENCİ MAIL: mdesenturk@gmail.com

DÖNEM: 2

GRUP NO: 1

Fonksiyon Tanımlamaları:

1- Linkli liste yapısında prev, next, left, right nodları adres ve value bilgileri saklandı. prev nodu insertion sort için kullanıldı.
2- create_list fonksiyonu verilen harf değerinden yeni bir linkli liste elemanı oluşturmak için oluşturuldu.
3- add_char_to list fonksiyonu verilen linkli listeye eğer verilen harf listede yoksa yeni eleman olarak ekleyen, listede varsa elemanın frekansını arttıran fonksiyon.
4- insertion_sort fonksiyonu verilen linkli listeyi frekans değerine göre insertion sort yapan fonskiyon.
5- convert_huffman verilen listenin ilk iki elemanını left ve right olarak alan ve frekans değerini ise bu elemanların toplamı olacak şekilde bir alan bir root elemanı oluşturup bu elemanı linkli listenin sıralamasını bozmayacak şekilde uygun yere yerleştirip bu işlemi liste boşalana dek tekrarlayarak listenin ağaca çevrilmesini sağlayan fonksiyon.
6- tree_height fonksiyonu verilen binary treenin yüksekliğini rekürsif olarak hesaplayan fonksiyon olarak kullanıldı.
7- print_level verilen linkli listenin istenilen sırasını eğer eleman varsa frekans ve harf değerini bastıran eğer o nodda eleman yolsa boşluk değerini bastıran fonksiyon.
8- print_huffman verilen treenin uzunluğunu tree_height fonksiyonu ile hesaplayıp her tree sırasını print_level ile yazdıran fonksiyon.

Algoritma:

1- Kullanıcıdan huffman tree'si oluşturulacak metin klavyeden alındı.
2- Alınan metnin her karakteri için add_char_to_list fonksiyonu çağırılarak linkli liste içerisinde eğer verilen karakter listede varsa o liste elemanının frekansını bir arttıracak, eğer listede yoksa o karakter için yeni eleman oluşturup listenin sonuna eklenecek şekilde linkli liste yapısı düzenlendi.
3- Bu liste yapısı insertion_sort fonksiyonu ile insertion_sort kurallarına göre sıralandı. Her noddan önceki noda prev nodu ile erişip onla karşılaştırmak gerektiğinden prev nodu oluşturuldu.
4- Sıralanan liste convert_huffman fonksiyonu ile huffman treeye çevrildi. Bu işlem için her seferde listenin ilk iki elemanı sırası ile left ve right node olacak şekilde yeni bir root nodu oluşturuldu. Oluşturulan root nodunun frekans değeri yine bu iki nodun frekanslarının toplamı olacak şekilde ayarlandı. Linkli listenin son 2 elemanı kalıncaya kadar bu

fonksiyon sürdürüldü. Son 2 eleman kaldığında ise elemanlardan küçük olanı left büyük olanı right node olacak şekilde ana düğüm oluşturuldu. Bu şekilde linkli liste yapısı binary treeye çevrilmiş oldu.

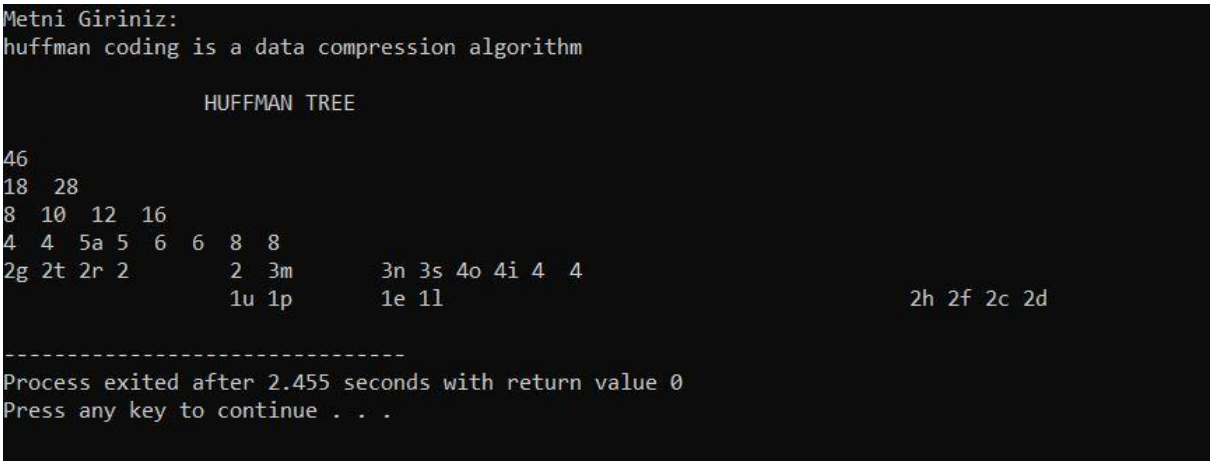
5- Oluşturulan binary treenin yazdırılması için print_huffman fonksiyonu çağrıldı. Bu fonksiyon ilk olarak tree_height fonksiyonu ile treenin uzunluğunu buldu. Bu işlemde rekürsif olarak her nodun sağ ve sol derinliği aşağıdan yukarı şekilde karşılaştırılarak ana noda kadar ulaşıldı. Bu şekilde en uzun dal bulunmuş oldu.

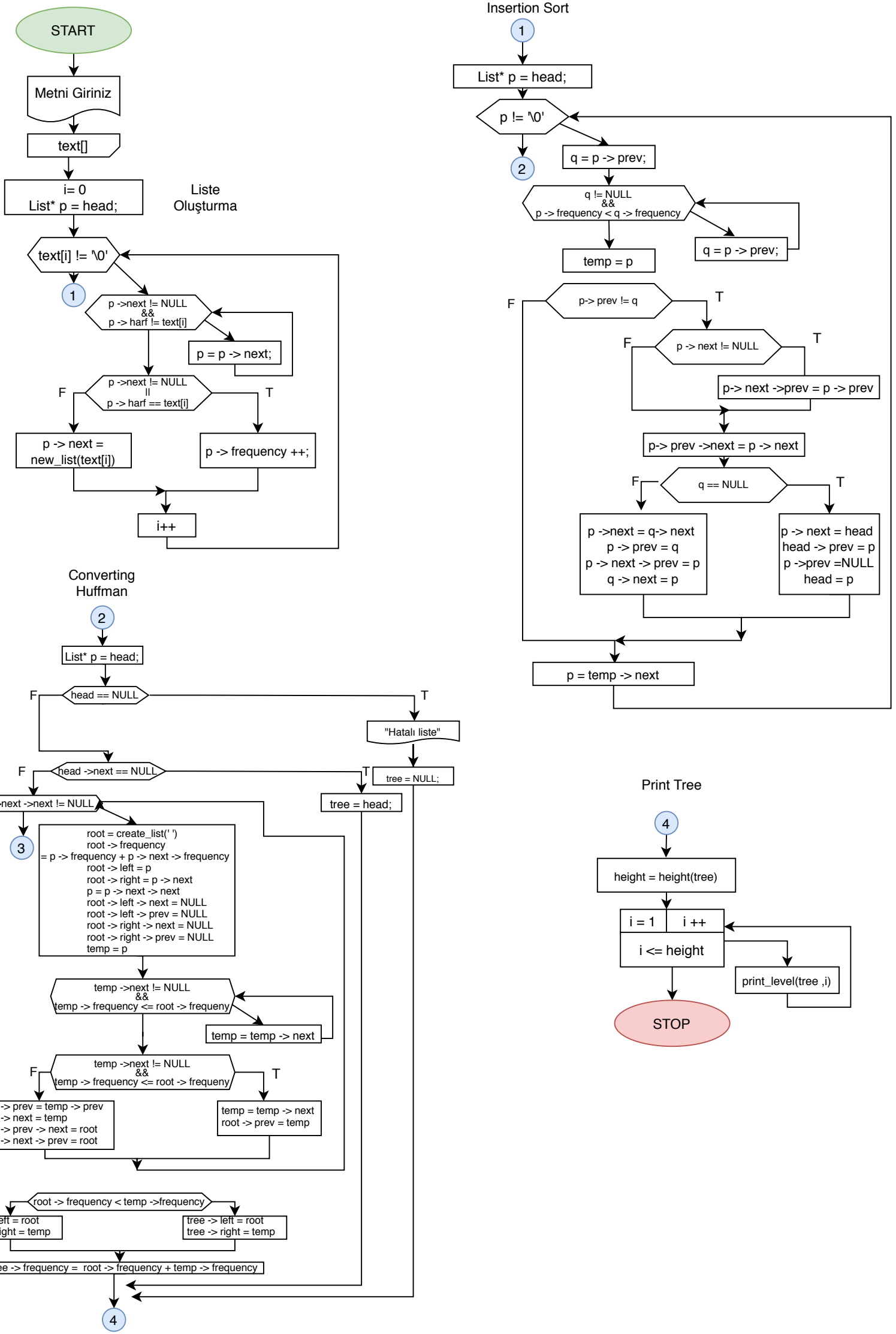
6- En uzun dal bulunduktan sonra bir for döngüsü içerisinde listenin her sırasını rekürsif olarak yazdıran print_level fonksiyonu çağrıldı. Bu fonksiyon sonucunda huffman treenin her dalı boş dallar için boşluk karakteri dolu dallar için ise dalın frekansı ve eğer içerdiği harf varsa harf değeri yazdırıldı. Harf değeri olmayan dallar için oluşturulurken boşluk karakteri ile oluşturulduğundan onların harf değerleri için boşluk yazdırılmış oldu.

Ekran Görüntüleri:

Input = huffman coding is a data compression algorithm

Output:




START
Metni Giriniz
text[]
i= 0
List* p = head;
Liste Oluşturma
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++
Converting Huffman
2
List* p = head;
head == NULL
"Hatalı liste"
head ->next == NULL
tree = NULL;
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
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
Insertion Sort
1
List* p = head;
p != '\0'
2
q = p -> prev;
q != NULL && p -> frequency < q -> frequency
q = p -> prev;
temp = p
p-> prev != q
p -> next != NULL
p-> next ->prev = p -> prev
p-> prev ->next = p -> next
q == NULL
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
Print Tree
4
height = height(tree)
i = 1
i ++
i <= height
print_level(tree ,i)
STOP