06.04.2020

YILDIZ TEKNİK ÜNİVERSİTESİ

ELEKTRİK-ELEKTRONİK FAKÜLTESİ

BİLGİSAYAR MÜHENDİSLİĞİ BÖLÜMÜ

BLM2512 VERİ YAPILARI VE ALGORİTMALAR 2.ÖDEV RAPORU

AHMET SAİD SAĞLAM

17011501

KONU

Huffman Ağacı Oluşturma

Verilen ödevde kayıpsız veri sıkıştırma yöntemlerinden Huffman Kodlamanın, Huffman Ağacı oluşturma aşamasına ait algoritmayı tasarlama ve gerçekleme aşamalarının yapılması istenmiştir.

İlk olarak sıkıştırılması istenen yazı kullanıcıdan elle veya dosya vasıtasıyla alınmış ve bu yazıdaki her harfin kullanım sıklığı hesaplanmıştır. Harfler ve kullanım sıklıkları single linked list yapısında saklanmıştır.

Sonrasında harflerin kullanım sıklıklarına göre düğümler insertion sort algoritmasıyla küçükten büyüğe sıralanmıştır.

Devam eden süreçte sıralı linkli liste yapısından dokümanda bahsedildiği şekilde Huffman ağacı oluşturulmuştur.

Son olarak oluşturulan bu Huffman ağacı ekrana yazdırılmış ve kullanıcın isteğine göre bellek alanı serbest bırakılarak program sonlandırılmıştır.

ÇÖZÜM

Kütüphane Eklenmesi ve Makro Atanması

```
1   /*
2   @file
3   BLM2512 2019-2020 BAHAR ODEV-2
4   Bu programda Single Linked List yapısı kullanılarak bir Huffman Tree tasarımı yapılmıştır.
5
6   @author
7   İsim: Ahmet Said SAĞLAM
8   Öğrenci No: 17011501
9   Tarih: 06.04.2020
10  E-Mail: l1117501@std.yildiz.edu.tr
11  Compiler: TDM-GCC 4.9.2 64 bit-Release
12  IDE: DEV-C++ (version 5.11)
13  İşletim Sistemi: Windows 10 Pro 64 bit
14  */
15  #include <stdio.h>
16  #include <stdlib.h>
17  #include <string.h>
18  #include <conio.h>
19  #include <stdbool.h>
20  #define LENGTH 50       //kullancidan elle alinabilecek maksimum string uzunlugu
```

Kodun ilk kısmında gerekli olabilecek kütüphaneler eklenmiş, daha sonra kod içerisinde gerekli bir makro atanmış ve programın geliştiricisi ve geliştirilip çalıştırıldığı ortam hakkında bilgiler verilmiştir.

Linkli Liste ve Ağaç Düğümü Yapı Tanımı

```
22  //aynı zamanda hem single linked list hem de tree olabilecek yapi tanimi
23  typedef struct node {
24      struct node* next;      //linkli listedeki sonraki eleman
25      struct node* right;     //agacin sag yapragi
26      struct node* left;      //agacin sol yapragi
27      int count;              //karakterin kac kez tekrar ettigini tutan degisken
28      char character;         //karakteri tutan degisken
29  } node;
30
```

Öncelikle problemi çözmek için ödev dokümantasyonunda belirtildiği üzere hem single linked list hem de tree oluşturabilecek "node" isimli bir yapı oluşturulmuştur.

Bu yapıda linkli listede bir sonraki düğümü işaret eden next isimli ve ağaçta ağacın sağ sol dallarını işaret eden left ve right isimli pointerlar, her node için saklanması gereken karakteri tutan "character" isimli char tipi değişken ve bu değişkenin frekansını tutan "count" isimli int tipi bir değişken tanımlanmıştır.

Linkli Listeye Head Node Ekleme Fonksiyonu

```
31  //linked list basa eleman ekleme fonksiyonu
32  node* basaEkle(node* head, char val) {
33      node* newnode;
34      newnode = (node*) malloc(sizeof(node));
35      newnode->character = val;
36      newnode->count = 1;
37      newnode->next = head;
38      newnode->left = NULL;
39      newnode->right = NULL;
40      head = newnode;
41      return head;
42  }
```

basaEkle fonksiyonu çağırıldığında yeni bir node oluşturur ve bu node'u linkli listenin head node'u olarak ayarlar. Ayrıca oluşturduğu node'a kullanıcıdan veya dosyadan alınan karakteri vererek count değerini "1" olarak ayarlar. Böylece sonradan aynı karakter girildiğinde count değeri bir arttırılır ve gerekli kontroller yapılabilir hale gelir. Ayrıca node aynı zamanda bir ağacın düğümü de olacağından ilk etapta oluşturulan bu yeni node'un left ve right işaretçileri null olarak ayarlanır.

Linkli Listeden Ağaç Oluşturma Fonksiyonları

```
44  //linkli listedeki count değerlerine göre araya elemen ekleme fonksiyonu
45  //agac olustruruken kullanilacagi icin sag ve sol node'lar ayarlanir
46  node* arayaEkle(node* head, node* temp, int toplam) {      //temp düğümü linkli listeye kendisinden sonra araya eleman eklenecek düğümdür.
47      node* newnode;
48      newnode = (node*) malloc(sizeof(node));
49      newnode->count = toplam;
50      newnode->character = 254;
51      newnode->next = temp->next;
52      temp->next = newnode;
53      newnode->left = head;
54      newnode->right = head->next;
55      return head;
56  }
```

arayaEkle fonksiyonu hazır bir linkli liste yapısından Huffman Tree oluşturmak amacıyla kullanılan fonksiyonlardan birisidir. Ödev dokümanında bahsedildiği üzere count değerleri küçükten büyüğe sıralı linkli listenin, sıralı iki node'unun count değerleri toplamı eğer listenin son veya daha önceki bir elemanının count değerinden küçükse oluşturulacak yeni node listede nodelar arasına eklenmelidir. Bu ihtiyaçtan dolayı bu fonksiyonda yeni bir node oluşturulur ve linkli listenin uygun yerine eklenir. Ayrıca yine dokümanda bahsedildiği üzere oluşturulan yeni node'un left pointer'ı head node, right pointer'ı ise head node'un next'i olacak şekilde ayarlanır. Bu şekilde ayarlanmasının nedeni toplanan iki count değerinin head ve head node'un next'ine ait olmasındandır çünkü linkli listede o an halihazırda bulunan en küçük iki count değerleri bu iki node'a aittir. Tüm bunlarla beraber oluşturulan bu yeni node'un count değeri diğer iki node'un toplanan count değerine eşitlenir ve son olarak oluşturulan bu yeni node'un character değişkenine ASCII 254 değeri atanır. Bu değerde bir kare işareti vardır ve yazdırıldığında ekranda bu şekilde gözükür. Bunun yapılmasının nedeni karakter değeri olmayan node'lar ekrana yazdırıldığında düzenli gözükmesi ve boşluk tuşunun karakteri ile karıştırılmamasıdır.

```
58  //linkli listedeki count değerlerine göre sona eleman ekleme fonksiyonu
59  //agac olustruruken kullanilacagi icin sag ve sol node'lar ayarlanir
60  node* sonaEkle(node* head, node* curr, int toplam) {        //curr = linkli listenin mevcut haldeki son düğümü
61      node* newnode;
62      newnode = (node*) malloc(sizeof(node));
63      newnode->count = toplam;
64      newnode->character = 254;
65      newnode->next = NULL;
66      curr->next = newnode;
67      newnode->left = head;
68      newnode->right = head->next;
69      return head;
70  }
```

sonaEkle fonksiyonu temel olarak arayaEkle fonksiyonu ile tamamen aynıdır. Tek farkları sonaEkle fonksiyonunda ekleme işlemi arayaEkle fonksiyonunda olduğu gibi linkli listede node'lar arasına değil de linkli listenin son node'u olarak ekleme şeklinde gerçekleşmesidir.

```
72  //linkli listenin bastan ilk iki elemanini silen fonksiyon (silme isleminde node'lar free edilmez cunku daha sonradan agacin dallari olacaklardir.)
73  node* bastanSilme(node* head) {
74      node* curr;
75      curr = head;            //ilk durumdaki listenin bastan ilk elemanını silme asamasi
76      head = head->next;
77      curr = head;            //ilk durumdaki listenin bastan ikinci elemanını silme asamasi
78      head = head->next;
79      return head;
80  }
```

bastanSilme fonksiyonu linkli listeden ağaç oluşturulurken head node ve head node'un next'inin bir dal olarak işaretlenmesinin ardından bu iki node'u linkli liste yapısından kaldırır.

Okunan Karakter Kontrolü Fonksiyonu

```
82  //linkli listeye disaridan gelen karakter mevcutta var mi diye kontrol eden fonksiyon
83  node* arama(node* head, char val) {
84      int bulgu = 1;
85      node* curr = head;
86      node* temp;
87      while(bulgu != 0 && curr->next !=NULL) {
88          if(curr->character == val){
89              bulgu = 0;
90          }
91          temp = curr;
92          curr = curr->next;
93      }
94      if(bulgu == 0) {
95          return temp;        //bas node veya ara nodelardan biri esit ise disariya adresi dondurulur
96      }
97      else {
98          if(curr->character == val){
99              return curr;    //son eleman esit ise disariya son elemanin adresi dondurulur
100         }
101         return NULL;        //mevcut degilse NULL dondurulur
102     }
103 }
```

arama fonksiyonu kullanıcıdan alınan veya dosyadan alınan bir string'den karakter okuma ve linkli liste oluşturma işlemi yapıldığı sürece sürekli arka planda çalışır. Fonksiyonun amacı eğer okunan karakter daha önce okunanlardan farklı değil ise fonksiyon dışarıya null döndürür. Böylece o karaktere özel yeni bir node oluşturulur. Karakter daha önce okundu ise de o node zaten oluşturulmuş demektir ve dışarıya o node'un adresi döndürülür ve böylece ilgili node'un count değeri bir arttırılabilir hale gelir.

İnsertion Sort Algoritması

```
107  //insertion sort yontemiyle linkli listeyi kucukten buyuge siralayan fonksiyon
108  void insertionSort(node** head) {
109      node* sirali = NULL;
110      node* current = *head;
111      while(current != NULL) {
112          node* next = current->next;
113          siraliEkle(&sirali, current);
114          current = next;
115      }
116      *head = sirali;         //head isaretcisi yeni linkli listenin head node'unu gosterecek sekilde ayarlanir.
117  }
```

insertionSort fonksiyonu linkli listeyi baştan itibaren okur ve siraliEkle fonksiyonu yardımıyla node'ların adresleri aracılığıyla, count değerleri küçükten büyüğe sıralı yeni bir linkli liste oluşturur. Sonrasında ise head pointer'ı oluşturulan yeni linkli listenin head node'unu gösterecek şekilde ayarlar ve linkli listeyi insertion sort yöntemiyle sıralamış olur.

```
119  //insertionSort fonksiyonu icinde cagirilarak linkli listeyi yeniden olusturarak siralar
120  void siraliEkle(node** head, node* newnode) {
121      node* current;
122      if(*head == NULL || (*head)->count >= newnode->count) { //ilk durumda head  null oldugunda veya eklencek node'un count degeri head node'unkinden daha kucukse
123          newnode->next = *head;
124          *head = newnode;
125      }
126      else {
127          current = *head;
128          while(current->next != NULL && current->next->count < newnode->count) { //yeni eklenecek node'un count degeri, listedeki ilk olarak hangi node'dan daha b
129              current = current->next;
130          }
131          newnode->next = current->next;
132          current->next = newnode;
133      }
134  }
```

siraliEkle fonksiyonu hazır linkli listeden gelen node'un count değerine göre küçükten büyüğe sıralı olacak şekilde, gelen node'u yeni linkli listenin uygun yerine yerleştirir.

Bu iki fonksiyonda yeni linkli liste oluşturarak sıralama işlemi, yeniden hafızada yer açarak değil de olan listenin adreslerini yeniden sıralayarak yapılır.

Linkli Listeyi Ekrana Yazdırma Fonksiyonu

```
136  //linkli listeyi ekrana yazdiran fonksiyon
137  void printList(node* head) {
138      node* temp = head;
139      while (temp->next != NULL) {
140          printf("%c,%d -> ",temp->character,temp->count);
141          temp = temp->next;
142      }
143      printf("%c,%d",temp->character,temp->count);
144
145  }
```

printList fonksiyonu head node'dan başlayarak son node'a doğru tüm node'ları, içerdiği character ve count değerleriyle birlikte sırayla ekrana yazdırır.

Binary Tree Yükseklik Hesaplama Fonksiyonu

```
147  //huffman tree'nin yuksekligini hesaplayan fonksiyon
148  int height(node* head) {
149      if(head == NULL) {
150          return 0;
151      }
152      else{
153          int leftDepth = height(head->left);
154          int rightDepth = height(head->right);
155          if(leftDepth > rightDepth) {
156              return (leftDepth+1);
157          }
158          else {
159              return (rightDepth+1);
160          }
161      }
162  }
```

height fonksiyonu Huffman Tree'nin yüksekliğini hesaplayan fonksiyondur. Head node'dan başlayarak önce sol dallara uğrayarak ağacın en alt seviyesine iner ve burdan itibaren sol ve sağ dallara göre hangisi mevcutsa onların yükseklik değerlerini 1 arttırarak yukarı çıkar. En sonunda fonksiyondan dönen değer ağacın yüksekliği olur.

Ağacın İlgili Seviyesini Ekrana Yazdıran Fonksiyon

```
164  //agacin ilgili seviyesini ekrana yazdiran fonksiyon
165  void printLevel(node* head, int level) {
166      if(head == NULL) {
167          printf(" ");
168          return;
169      }
170      if(level == 1) {
171          printf("%d'%c' ",head->count,head->character);
172      }
173      else if(level > 1) {
174          printLevel(head->left, level-1);
175          printLevel(head->right, level-1);
176
177      }
178  }
```

printLevel fonksiyonu Huffman Tree'nin verilen seviyesini ekrana yazdırır. Önce sol dalları dolaşır ve ilgili seviyeye geldiğinde varsa sol node'u yoksa sağ node'u ikisi de mevcutsa önce sol sonra sağ node'u ekrana yazdırır ve yukarı çıkar. Yukarıdaki düğüm herhangi bir sağ dala sahipse yeniden aşağı iner ve yine aynı işlemleri takip eder. Böylece ağacın fonksiyona verilen seviyesi ekrana yazdırılır.

Binary Tree Ekrana Yazdırma Fonksiyonu

```
180  //agaci butun seviyeleriyle birlikte kokten yapraklara kadar ekrana yazdiran fonksiyon
181  void printTree(node* head) {
182      int yukseklik = height(head);
183      int i;
184      for (i=1; i<=yukseklik; i++) {
185          printLevel(head,i);
186          printf("\n");
187      }
188  }
```

printTree fonksiyonu, printLevel ve height fonksiyonları sayesinde Huffman Tree'yi bütün seviyeleriyle kökten yapraklara kadar sırasıyla ekrana yazdırır.

İnput Buffer Temizleme Fonksiyonu

```c
//gets() fonksiyonu kullanılmadan önce eger scanf kullanıldıysa gets() duzgun calismayacagi icin input buffer'i temizlemmek icin gerekli fonksiyon
int clear_input_buffer(void) {
    int ch;
    while (((ch = getchar()) != EOF) && (ch != '\n'));
    return ch;
}
```

clear_input_buffer fonksiyonu, kodun main fonksiyonu içinde scanf() fonksiyonundan sonra gets() fonksiyonu kullanılacağından input buffer'ı temizlemek için kullanılır. Aksi taktirde gets() fonksiyonu input buffer'da scanf() fonksiyonundan kalan ENTER girişini ilk olarak okuyacak ve istenen sonucu vermeyecektir.

Binary Tree Silme Fonksiyonu

```c
//post-traverse kullanarak agaci memory'den temizleyen fonksiyon
void deleteTree(node* head) {
    if(head == NULL){
        return;
    }
    deleteTree(head->left);
    deleteTree(head->right);
    free(head);
}
```

deleteTree fonksiyonu post-traverse yöntemi kullanarak ağacı bütün düğümleriyle birlikte temizler ve bellek alanını serbest bırakır.

Ana Fonksiyon Başlangıcı ve Tanımlamalar

```c
int main() {
    FILE *fp;               //file pointer
    int i;                  //for dongusu icin indis tanimi
    int cls;                //program sonlanirken agaci temizlemek veya oldugu gibi birakmak durumunu kontrol eden degisken
    int toplam;             //linkli listenin dugumlerinden agac olustururken, iki dugumun count degerlerinin toplamini tutup yeni node'a count degeri
    int tur;                //girdinin dosyadan mi yoksa kullanicidan elle mi alinacagini kontrol eden degisken
    char string[LENGTH];    //girdinin kullanicidan elle alinmasi durumunda bu veriyi saklayan karakter dizisi
    char val;               //linkli listenin her bir dugumunde bulunan karakteri disaridan veren degisken
    char filename[50];      //girdinin dosyadan okunmasi durumunda dosya ismini tutacak olan karakter dizisi
    node* head = NULL;      //linkli listenin ilk dugumu ilklendirilir
    node* curr;             //
    node* prev;             //linkli listede ve agac olustururken kullaniacak olan dugum isaret eden gecici degiskenler
    node* temp;             //
    printf("Metni dosyadan almak icin 1'e, elle girmek icin 2'ye basiniz.\n");
    scanf("%d",&tur);
```

main fonksiyonunun içinde öncelikle kullanılacak değişkenlerin, dizilerin ve pointerların tanımı yapılır. Ardından kullanıcıya metnin elle mi girileceği yoksa dosyadan mı alınacağı sorulur. Kullanıcının seçimine göre program akışı devam eder.

Verinin Dosyadan Alınması

```
222     if(tur == 1) {                          //metin dosyadan alinir
223         printf("Lutfen acmak istediginiz dosyanin ismini uzantisiyla birlikte giriniz.\n");
224         scanf("%s",filename);
225         if((fp = fopen(filename,"r")) == NULL) {
226             printf("Dosya acilamadi!\n");
227             return 0;
228         }
229         else {
230             val = fgetc(fp);
231             head = basaEkle(head,val);
232             while(!feof(fp)) {              //dosyadaki string karakter karakter okunur ve linkli liste olusturulur
233                 val = fgetc(fp);
234                 if(val != EOF && val != 10) {
235                     temp = arama(head,val);
236                     if(temp == NULL) {
237                         head = basaEkle(head,val);
238                     }
239                     else {
240                         temp->count++;
241                     }
242                 }
243             }
244             fclose(fp);
245         }
246     }
```

Kullanıcı metni dosya aracılığıyla almayı seçtiğinde program kullanıcıya dosya ismini sorar ve eğer mevcutsa o dosyayı açar. Eğer dosya mevcut değilse kullanıcıya uyarı verilir. Dosya açıldığında içerisindeki string karakter karakter okunur ve her karaktere özel bir node oluşturularak linkli listeye eklenir. Karakter kendini tekrarlarsa zaten mevcut olan node'un count değeri 1 arttırılır. Böylece bütün karakterler okunmuş ve linkli liste hazırlanmış olur.

Verinin Kullanıcıdan Elle Alınması

```
247     else if(tur == 2) {                     //metin kullanicidan elle alinir
248         printf("String giriniz.\n");
249         clear_input_buffer();               //input buffer temizlenir
250         gets(string);
251         val = string[0];
252         head = basaEkle(head,val);
253         for(i=1; i<strlen(string); i++){        //elle girilen string karakter karakter okunur ve linkli liste olusturulur
254             val = string[i];
255             temp = arama(head,val);
256             if(temp == NULL){
257                 head = basaEkle(head,val);
258             }
259             else {
260                 temp->count++;
261             }
262         }
263     }
```

Kullanıcı metni elle girmeyi seçtiğinde ilk olarak input buffer temizlenir. Daha sonra gets() fonksiyonu aracılığıyla kullanıcıdan bir string alınır ve aynı dosyadan okumada olduğu gibi string karakter karakter okunur ve linkli liste oluşturulur.

```
264     else {
265         printf("Hatali Giris Yaptiniz!\nProgram Sonlandiriliyor...\n");  //veri alma seklini belirlemede hatali kullanici girisi
266         return 0;                                                        //program sonlanir
267     }
```

Kullanıcı veri giriş tipini belirlerken hatalı bir giriş yaparsa program kullanıcıyı uyarır ve program sonlanır.

Linkli Listenin Ekrana Yazdırılması

```
268     printf("\nLinkli Liste:\n\n");            //
269     printList(head);                         //
270     printf("\n\n");                          // Linkli liste önce oldugu gibi sonra da sirali sekilde ekrana yazdirilir.
271     insertionSort(&head);                    //
272     printf("Sirali Linkli Liste:\n\n");      //
273     printList(head);                         //
274
```

Sonrasında oluşturulan linkli liste önce olduğu şekliyle daha sonrasında ise node'ların count değerlerine göre insertion sort algoritmasıyla küçükten büyüğe sıralanarak ekrana yazdırılır.

Linkli Listeden Huffman Tree Oluşturulması

```
275     //linkli listeden agaci olusturan blok...
276     while(head->next != NULL) {
277         curr = head;
278         toplam = head->count + head->next->count;
279         while(toplam >= curr->count && curr->next != NULL) {
280             prev = curr;
281             curr = curr->next;
282         }
283         if(curr->next == NULL) {
284             head = sonaEkle(head,curr,toplam);
285             head = bastanSilme(head);
286         }
287         else {
288             head = arayaEkle(head,prev,toplam);
289             head = bastanSilme(head);
290         }
291     }
292     //...linkli listeden agaci olusturan blok
```

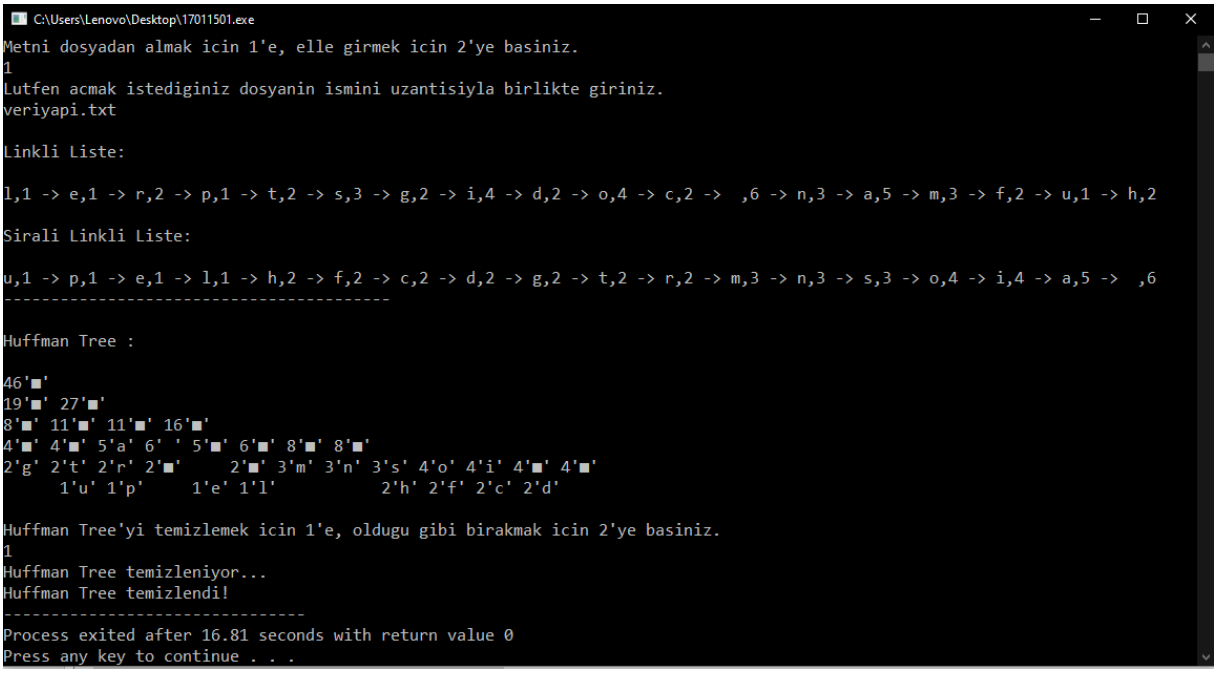
Sıralanan linkli listeden Huffman Tree oluşturulması bu kod bloğu sayesinde olur. Öncelikle raporun önceki kısımlarında bahsedildiği üzere head node ve head node'un next'inin count değerleri toplanır ve toplam isimli değişkende tutulur. Bu toplam değerine göre yeni bir node oluşturulur ve oluşturulan bu yeni node'un left ve right pointerları head node'a göre atanır ve hemen sonrasında oluşturulan yeni node, linkli listenin count değerlerine göre uygun yerine yerleştirilir. Ardından linkli listenin ilk iki node'u listeden kaldırılır. Bu işlem listede tek bir node kalıncaya kadar devam eder. Sona gelindiğinde Huffman Tree oluşmuş olur.

Huffman Ağacının Ekrana Yazdırılması ve Temizleme İzni İstenmesi

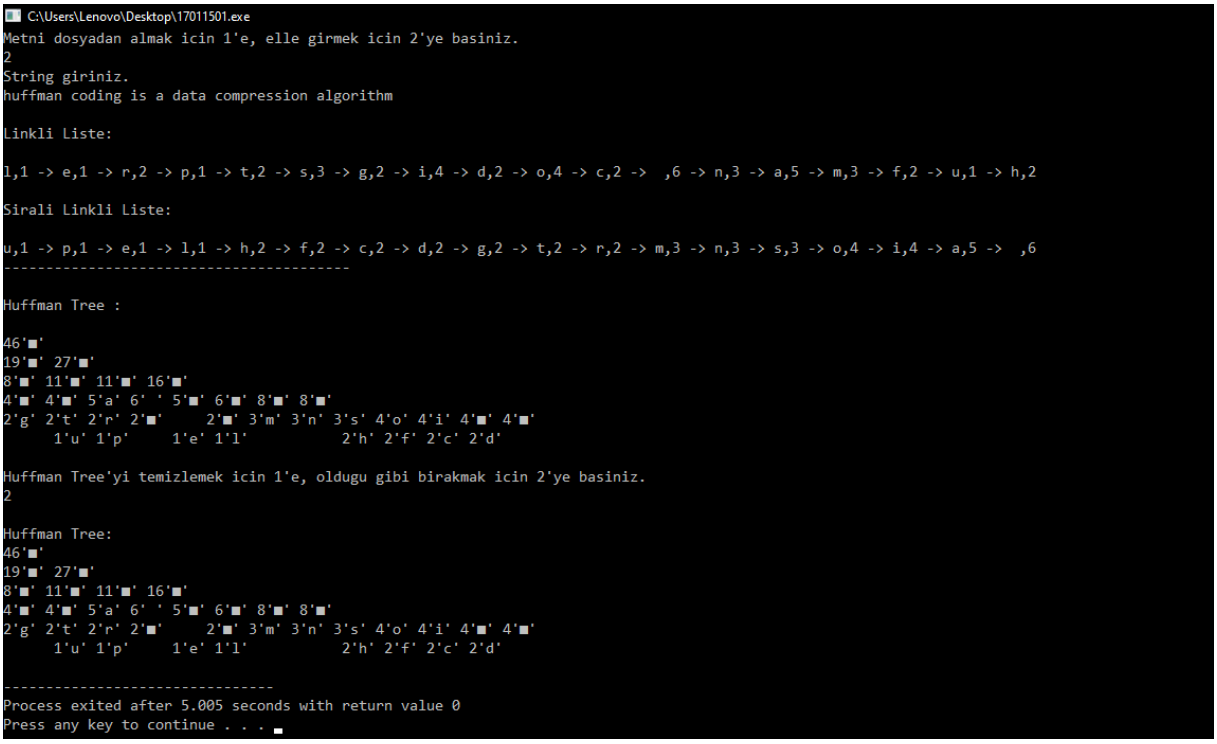
```
294     printf("\n----------------------------------------\n\nHuffman Tree : \n\n");
295     printTree(head);
296     printf("\nHuffman Tree'yi temizlemek icin 1'e, oldugu gibi birakmak icin 2'ye basiniz.\n");
297     scanf("%d",&cls);
298     if(cls == 1) {
299         printf("Huffman Tree temizleniyor...\n");
300         deleteTree(head);
301         printf("Huffman Tree temizlendi!");
302     }
303     else {
304         printf("\nHuffman Tree:\n");
305         printTree(head);
306     }
307     return 0;
308 }
```

Kodun son kısmında Huffman Tree ekrana yazdırılır ve kullanıcıya ağacı temizlemek isteyip istemediği sorulur. Alınan cevaba göre ağaç temizlenir veya temizlenmeden bırakılarak yeniden ekrana yazdırılır ve program sonlanır.

PROGRAM ÇIKTILARI



Metnin Dosyadan Alınması ve Ağacın Temizlenmesi



Metnin Kullanıcıdan Elle Alınması ve Ağacın Temizlenmeden Bırakılması

SOURCE CODE

```
1.  /*
2.  @file
3.  BLM2512 2019-2020 BAHAR ODEV-2
4.  Bu programda Single Linked List yapısı kullanılarak bir Huffman Tree tasarımı yapılm
    ıştır.
5.
6.  @author
7.  İsim: Ahmet Said SAĞLAM
8.  Öğrenci No: 17011501
9.  Tarih: 06.04.2020
10. E-Mail: l1117501@std.yildiz.edu.tr
11. Compiler: TDM-GCC 4.9.2 64 bit-Release
12. IDE: DEV-C++ (version 5.11)
13. İşletim Sistemi: Windows 10 Pro 64 bit
14. */
15. #include <stdio.h>
16. #include <stdlib.h>
17. #include <string.h>
18. #include <conio.h>
19. #include <stdbool.h>
20. #define LENGTH 50       //kullancidan elle alinabilecek maksimum string uzunlugu
21.
22. //aynı zamanda hem single linked list hem de tree olabilecek yapi tanimi
23. typedef struct node {
24.     struct node* next;      //linkli listedeki sonraki eleman
25.     struct node* right;     //agacin sag yapragi
26.     struct node* left;      //agacin sol yapragi
27.     int count;              //karakterin kac kez tekrar ettigini tutan degisken
28.     char character;         //karakteri tutan degisken
29. } node;
30.
31. //linked list basa eleman ekleme fonksiyonu
32. node* basaEkle(node* head, char val) {
33.     node* newnode;
34.     newnode = (node*) malloc(sizeof(node));
35.     newnode->character = val;
36.     newnode->count = 1;
37.     newnode->next = head;
38.     newnode->left = NULL;
39.     newnode->right = NULL;
40.     head = newnode;
41.     return head;
42. }
43.
44. //linkli listedeki count değerlerine göre araya elemen ekleme fonksiyonu
45. //agac olustruruken kullanilacagi icin sag ve sol node'lar ayarlanir
46. node* arayaEkle(node* head, node* temp, int toplam) {       //temp düğümü linkli lis
    teye kendisinden sonra araya eleman eklenecek düğümdür.
47.     node* newnode;
48.     newnode = (node*) malloc(sizeof(node));
49.     newnode->count = toplam;
50.     newnode->character = 254;
51.     newnode->next = temp->next;
52.     temp->next = newnode;
53.     newnode->left = head;
54.     newnode->right = head->next;
55.     return head;
56. }
57.
58. //linkli listedeki count değerlerine göre sona eleman ekleme fonksiyonu
59. //agac olustruruken kullanilacagi icin sag ve sol node'lar ayarlanir
60. node* sonaEkle(node* head, node* curr, int toplam) {        //curr = linkli listenin
     mevcut haldeki son düğümü
```

```
61.     node* newnode;
62.     newnode = (node*) malloc(sizeof(node));
63.     newnode->count = toplam;
64.     newnode->character = 254;
65.     newnode->next = NULL;
66.     curr->next = newnode;
67.     newnode->left = head;
68.     newnode->right = head->next;
69.     return head;
70. }
71.
72. //linkli listenin bastan ilk iki elemanini silen fonksiyon (silme isleminde node'lar
     free edilmez cunku daha sonradan agacin dallari olacaklardir.)
73. node* bastanSilme(node* head) {
74.     node* curr;
75.     curr = head;            //ilk durumdaki listenin bastan ilk elemanını silme asam
    asi
76.     head = head->next;
77.     curr = head;            //ilk durumdaki listenin bastan ikinci elemanını silme a
    samasi
78.     head = head->next;
79.     return head;
80. }
81.
82. //linkli listeye disaridan gelen karakter mevcutta var mi diye kontrol eden fonksiyo
    n
83. node* arama(node* head, char val) {
84.     int bulgu = 1;
85.     node* curr = head;
86.     node* temp;
87.     while(bulgu != 0 && curr->next !=NULL) {
88.         if(curr->character == val){
89.             bulgu = 0;
90.         }
91.         temp = curr;
92.         curr = curr->next;
93.     }
94.     if(bulgu == 0) {
95.         return temp;        //bas node veya ara nodelardan biri esit ise disariya ad
    resi dondurulur
96.     }
97.     else {
98.         if(curr->character == val){
99.             return curr;    //son eleman esit ise disariya son elemanin adresi dondu
    rulur
100.            }
101.            return NULL;        //mevcut degilse NULL dondurulur
102.        }
103.    }
104.
105.    void siraliEkle(node**, node*);
106.
107.    //insertion sort yontemiyle linkli listeyi kucukten buyuge siralayan fonksiyo
    n
108.    void insertionSort(node** head) {
109.        node* sirali = NULL;
110.        node* current = *head;
111.        while(current != NULL) {
112.            node* next = current->next;
113.            siraliEkle(&sirali, current);
114.            current = next;
115.        }
116.        *head = sirali;         //head isaretcisi yeni linkli listenin head node'
    unu gosterecek sekilde ayarlanir.
117.    }
118.
```

```
119.       //insertionSort fonksiyonu icinde cagirilarak linkli listeyi yeniden olustura
   rak siralar
120.       void siraliEkle(node** head, node* newnode) {
121.           node* current;
122.           if(*head == NULL || (*head)->count >= newnode-
   >count) {    //ilk durumda head  null oldugunda veya eklencek node'un count degeri h
   ead node'unkinden daha kucukse basa eklenir.
123.               newnode->next = *head;
124.               *head = newnode;
125.           }
126.           else {
127.               current = *head;
128.               while(current->next != NULL && current->next->count < newnode-
   >count) {  //yeni eklenecek node'un count degeri, listedeki ilk olarak hangi node'da
   n daha buyukse bulmak icin while dongusu
129.                   current = current->next;
130.               }
131.               newnode->next = current->next;
132.               current->next = newnode;
133.           }
134.       }
135.
136.       //linkli listeyi ekrana yazdiran fonksiyon
137.       void printList(node* head) {
138.           node* temp = head;
139.           while (temp->next != NULL) {
140.               printf("%c,%d -> ",temp->character,temp->count);
141.               temp = temp->next;
142.           }
143.           printf("%c,%d",temp->character,temp->count);
144.
145.       }
146.
147.       //huffman tree'nin yuksekligini hesaplayan fonksiyon
148.       int height(node* head) {
149.           if(head == NULL) {
150.               return 0;
151.           }
152.           else{
153.               int leftDepth = height(head->left);
154.               int rightDepth = height(head->right);
155.               if(leftDepth > rightDepth) {
156.                   return (leftDepth+1);
157.               }
158.               else {
159.                   return (rightDepth+1);
160.               }
161.           }
162.       }
163.
164.       //agacin ilgili seviyesini ekrana yazdiran fonksiyon
165.       void printLevel(node* head, int level) {
166.           if(head == NULL) {
167.               printf(" ");
168.               return;
169.           }
170.           if(level == 1) {
171.               printf("%d'%c' ",head->count,head->character);
172.           }
173.           else if(level > 1) {
174.               printLevel(head->left, level-1);
175.               printLevel(head->right, level-1);
176.
177.           }
178.       }
179.
```

```
180.       //agaci butun seviyeleriyle birlikte kokten yapraklara kadar ekrana yazdiran
   fonksiyon
181.       void printTree(node* head) {
182.           int yukseklik = height(head);
183.           int i;
184.           for (i=1; i<=yukseklik; i++) {
185.               printLevel(head,i);
186.               printf("\n");
187.           }
188.       }
189.
190.       //gets() fonksiyonu kullanılmadan önce eger scanf kullanıldıysa gets() duzgun
    calismayacagi icin input buffer'i temizlemmek icin gerekli fonksiyon
191.       int clear_input_buffer(void) {
192.           int ch;
193.           while (((ch = getchar()) != EOF) && (ch != '\n'));
194.           return ch;
195.       }
196.
197.       //post-traverse kullanarak agaci memory'den temizleyen fonksiyon
198.       void deleteTree(node* head) {
199.           if(head == NULL){
200.               return;
201.           }
202.           deleteTree(head->left);
203.           deleteTree(head->right);
204.           free(head);
205.       }
206.
207.       int main() {
208.           FILE *fp;               //file pointer
209.           int i;                  //for dongusu icin indis tanimi
210.           int cls;                //program sonlanirken agaci temizlemek veya oldug
   u gibi birakmak durumunu kontrol eden degisken
211.           int toplam;             //linkli listenin dugumlerinden agac olustururken
   , iki dugumun count degerlerinin toplamini tutup yeni node'a count degeri olarak ata
   nmasini saglayan degisken
212.           int tur;                //girdinin dosyadan mi yoksa kullanicidan elle mi
    alinacagini kontrol eden degisken
213.           char string[LENGTH];    //girdinin kullanicidan elle alinmasi durumunda b
   u veriyi saklayan karakter dizisi
214.           char val;               //linkli listenin her bir dugumunde bulunan karak
   teri disaridan veren degisken
215.           char filename[50];      //girdinin dosyadan okunmasi durumunda dosya ismi
   ni tutacak olan karakter dizisi
216.           node* head = NULL;      //linkli listenin ilk dugumu ilklendirilir
217.           node* curr;             //
218.           node* prev;             //linkli listede ve agac olustururken kullaniacak
    olan dugum isaret eden gecici degiskenler
219.           node* temp;             //
220.           printf("Metni dosyadan almak icin 1'e, elle girmek icin 2'ye basiniz.\n")
   ;
221.           scanf("%d",&tur);
222.           if(tur == 1) {                          //metin dosyadan alinir
223.               printf("Lutfen acmak istediginiz dosyanin ismini uzantisiyla birlikte
    giriniz.\n");
224.               scanf("%s",filename);
225.               if((fp = fopen(filename,"r")) == NULL) {
226.                   printf("Dosya acilamadi!\n");
227.                   return 0;
228.               }
229.               else {
230.                   val = fgetc(fp);
231.                   head = basaEkle(head,val);
232.                   while(!feof(fp)) {              //dosyadaki string karakter karak
   ter okunur ve linkli liste olusturulur
```

```
                        val = fgetc(fp);
                        if(val != EOF && val != 10) {
                            temp = arama(head,val);
                            if(temp == NULL) {
                                head = basaEkle(head,val);
                            }
                            else {
                                temp->count++;
                            }
                        }
                    }
                    fclose(fp);
                }
            }
            else if(tur == 2) {                         //metin kullanicidan elle alinir
                printf("String giriniz.\n");
                clear_input_buffer();                   //input buffer temizlenir
                gets(string);
                val = string[0];
                head = basaEkle(head,val);
                for(i=1; i<strlen(string); i++){         //elle girilen string karakter karakter okunur ve linkli liste olusturulur
                    val = string[i];
                    temp = arama(head,val);
                    if(temp == NULL){
                        head = basaEkle(head,val);
                    }
                    else {
                        temp->count++;
                    }
                }
            }
            else {
                printf("Hatali Giris Yaptiniz!\nProgram Sonlandiriliyor...\n");  //veri alma seklini belirlemede hatali kullanici girisi
                return 0;                                                       //program sonlanir
            }
            printf("\nLinkli Liste:\n\n");              //
            printList(head);                            //
            printf("\n\n");                             // Linkli liste önce oldugu gibi sonra da sirali sekilde ekrana yazdirilir.
            insertionSort(&head);                       //
            printf("Sirali Linkli Liste:\n\n");         //
            printList(head);                            //

            //linkli listeden agaci olusturan blok...
            while(head->next != NULL) {
                curr = head;
                toplam = head->count + head->next->count;
                while(toplam >= curr->count && curr->next != NULL) {
                    prev = curr;
                    curr = curr->next;
                }
                if(curr->next == NULL) {
                    head = sonaEkle(head,curr,toplam);
                    head = bastanSilme(head);
                }
                else {
                    head = arayaEkle(head,prev,toplam);
                    head = bastanSilme(head);
                }
            }
            //...linkli listeden agaci olusturan blok

```

```
294.            printf("\n----------------------------------------
    \n\nHuffman Tree : \n\n");
295.            printTree(head);
296.            printf("\nHuffman Tree'yi temizlemek icin 1'e, oldugu gibi birakmak icin
    2'ye basiniz.\n");
297.            scanf("%d",&cls);
298.            if(cls == 1) {
299.                printf("Huffman Tree temizleniyor...\n");
300.                deleteTree(head);
301.                printf("Huffman Tree temizlendi!");
302.            }
303.            else {
304.                printf("\nHuffman Tree:\n");
305.                printTree(head);
306.            }
307.            return 0;
308.        }
```