Course Name: VERİ YAPILARI VE ALGORİTMALAR
Course Group: Group 1

Instructor Name: Doç. Dr. M.Elif KARSLIGİL

ÖDEV 2

Student Id: 16011706
Student Name and Surname: Duygu Erduran

Konu: Huffman Ağacı Oluşturma

Açıklama:

- File Dosya fonksiyonları
- String fonksiyonları
- Linkli Liste yapısı
- Struct : Node yapısı için oluşturuldu.
- Typedef tanımlaması
- While & For döngüsü
- If/Else/Else If Statements
- Kuyruk dizaynı

kullanıldı.

C Kodu:

```c
#include <stdio.h>
#include <windows.h>
#include <locale.h>
#include<math.h>
#define MAX 300

struct node {     //struct yapimizi olusturduk
      char harf;
      int frekans;
      struct node *next, *left, *right;
};
typedef struct node NODE;
//  fonksiyonlarimizi tanitiyoruz
struct node* dugumBul(struct node* , char );
NODE *linkli_liste_olusturma(NODE *, char );
NODE * insertion_sort(NODE *);
NODE* agac(NODE* );
void dosyayi_okuma(char *buffer, FILE *dosya);

void dosyayi_okuma(char *buffer, FILE *dosya){ // text dosyasini okuma
yapan fonksiyon
      char dosyaismi[20];
      int i = 0;
      printf("Dosya Ýsmi: ");
      scanf("%s",dosyaismi);
```

```
	dosya = fopen(dosyaismi, "r");  // dosyayi read konumda açar.
	if(dosya == NULL){                // dosya nullsa ekrana hata
mesaji yazar. ve program sonlanir.
		printf("Dosya açýlamadý.");
		exit(1);
	}
	fgets(buffer, MAX, dosya);   // dosyadan aldigi metni ekrana
yazar.
	printf("Metin: ");
	NODE *head = NULL;
	while(i<strlen(buffer)){
		head = linkli_liste_olusturma(head, buffer[i]);  //Her harfe
bir dugum olusturuyoruz
		printf("%c", buffer[i]);
		i++;
	}
	printf("\n");
	head = insertion_sort(head);
	head = agac(head);
	agaci_siralama(head);
}

NODE *linkli_liste_olusturma(NODE *head, char gelenHarf) {  // linkli
listeyi olusturur.
	NODE *temp = (NODE *)malloc(sizeof(NODE));
	NODE *temp2 = (NODE *)malloc(sizeof(NODE));

	if(head == NULL){
		temp -> frekans = 1;
		temp -> next = NULL;
		temp -> harf = gelenHarf;
		temp -> left = NULL;
		temp -> right = NULL;
		head = temp;
		return head;
	}else{
		temp2 = dugumBul(head, gelenHarf);
		if( temp2 == NULL){
			temp -> frekans = 1;
			temp -> next = NULL;
			temp -> harf = gelenHarf;
			temp->next = head;
				temp -> left = NULL;
		temp -> right = NULL;
			head = temp;
			return head;
```

```
            }
            else{
                  temp2->frekans++;
            }
      }
      return head;
}

struct node* dugumBul(struct node* head, char key) { // mantigi harf
aramak.
      struct node* dugumBul = NULL;   //eger listede  harf varsa dügümü
dönüyor yoksa null,
      while(head != NULL) // ve harf artirmak.eger listede  harf varsa
dügümü dönüyor yoksa null, ve harf artirmak.
            if(head -> harf != key)
                  head = head -> next;
            else {
                  dugumBul = head;
                  break;
      }
      return(dugumBul);
}

NODE* insertion_sort(NODE* head)   // linkli listeyi insert sort yapan
fonksiyon
{
    NODE *i, *j, *k;

    i = head;
    head = NULL;

    while(i != NULL)
    {
        k = i;
        i = i->next;
        if (head != NULL)
        {
            if(k->frekans > head->frekans)
            {
                j = head;
                while ((j->next != NULL) && (k->frekans> j->next-
>frekans))
                {
                    j = j->next;
                }
                k->next = j->next;
```

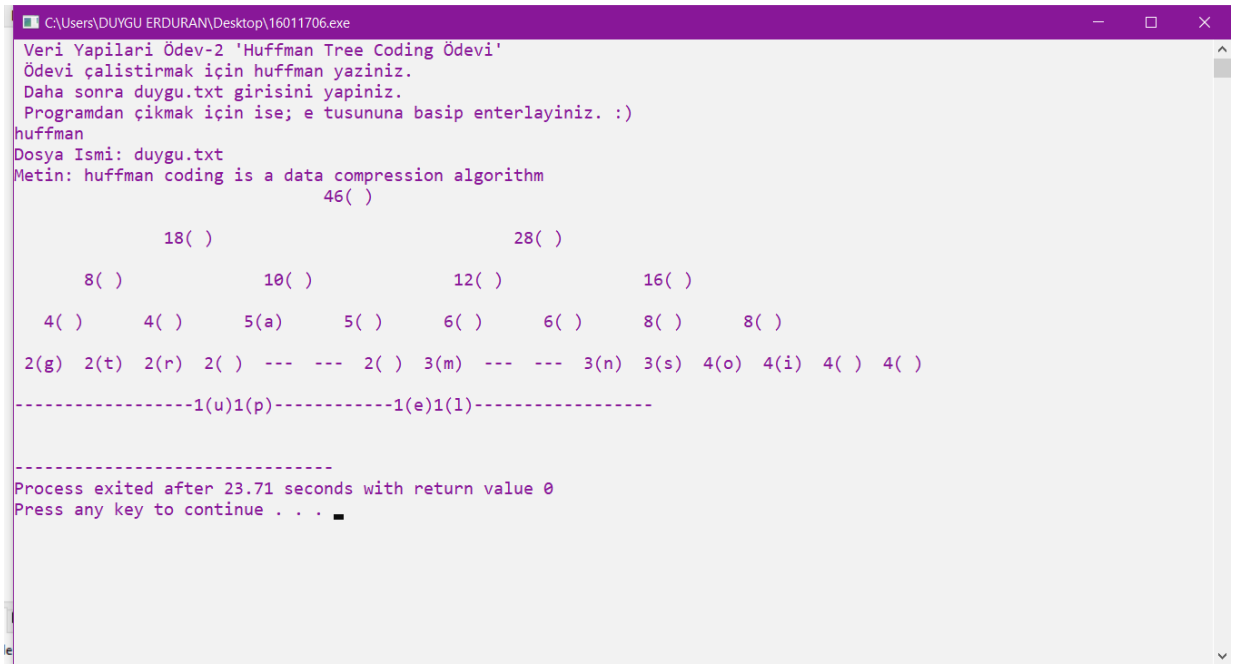
```
                j->next = k;
            }
            else
            {
                k->next = head;
                head = k ;
            }
        }
        else
        {
            k->next = NULL;
            head = k ;
        }
    }
    return head;
}

NODE* agac(NODE* head) { // nodelari agaca cevirmek icin olusturulan
fonksiyon

      NODE* current1 = head;
      NODE* current2 = head;
      while(current1->next != NULL){
            NODE* yeniDugum = (NODE*)malloc(sizeof(NODE));
            yeniDugum->left = NULL;
            yeniDugum->right = NULL;
            yeniDugum->next = NULL;
            current2 = current1;
            yeniDugum->frekans = current2->frekans+ current2->next-
>frekans;
            ekleme(head,yeniDugum);
            yeniDugum->right = current2->next;
            yeniDugum->left = current2;
            yeniDugum->harf = ' ';
            current1 = current1->next->next;
            head = current1;
      }
      return head;
}

void ekleme(NODE* head, NODE* yeniDugum){    // araya eleman ekler

            NODE* current = head;
            if(yeniDugum->frekans <= (head)->frekans) {
                  yeniDugum->next = head;
                  head = yeniDugum;
```

```
            }
            else{
                  while(current->next != NULL && current->next->frekans
<= yeniDugum->frekans) {
                        current= current->next;
                  }
                  yeniDugum->next = current->next;
                  current->next = yeniDugum;
            }
}

void agaci_siralama(NODE *head){  // agaci siralama

      if(head == NULL){  //eger agac bossa
            return;
      }
    //agacin seviyesi icin kuyrukla bir dizi olusturuyoruz
      NODE *q[100];
      int enq = 0, deq = 0;
      int bosluk = pow(2, agac_yukseklik(head) -1) -1; // her
karakterin bosluklarini yazdirmak icin
      int level = agac_yukseklik(head); //agacin yüksekligini hesaplar
      int i, j;
      q[enq++] = head;
      for(i = 0; i < level; i++){ //agacin  her seviyesinin olusmasini
saglar.
            for(j = 0; j < pow(2, i); j++){
                  NODE *current = q[deq++];
                  if(current != NULL){ // kuyruktaki sol ve sag
cocuklara bakiyoruz
                        q[enq++] = current->left;
                        q[enq++] = current->right;
                  }
                  else{
                        q[enq++] = NULL;
                        q[enq++] = NULL;
                  }
                  if(current != NULL){  // current null değilse,bosluk
fonksiyonunu cagirarak yazdirma yapar.
                  bosluk_yazdir(bosluk);
                  printf("%d(%c)", current->frekans, current->harf);
                  bosluk_yazdir(bosluk);
                  }
                  else{ // eger bossa tire isareti koyar agacta
                        bosluk_yazdir(bosluk);
                        printf("---");
```

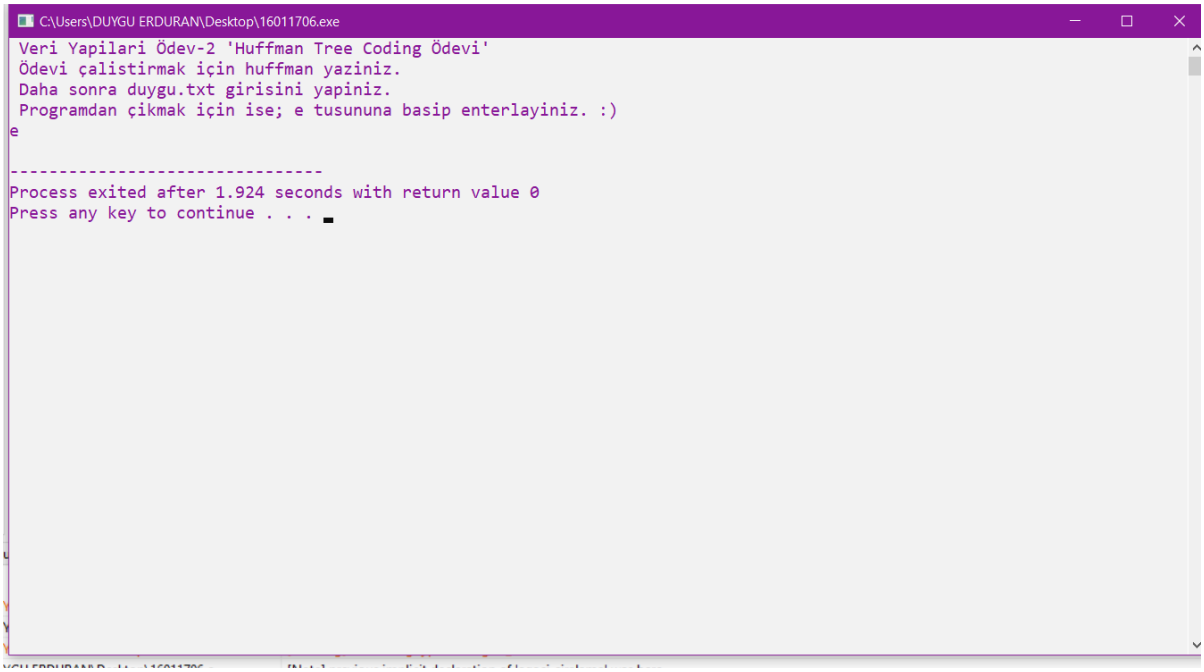
```
                    bosluk_yazdir(bosluk);
                }
            }
            bosluk = bosluk/2;
            printf("\n\n");
        }
}

int agac_yukseklik(NODE *head){ // agacin yüksekligini hesaplar
        if(head == NULL){
            return 0;
        }else{
            int sol_yukseklik = agac_yukseklik(head->left);
            int sag_yukseklik = agac_yukseklik(head->right);
            if(sol_yukseklik > sag_yukseklik){
                return sol_yukseklik+1;
            }else{
                return sag_yukseklik+1;
            }
        }
}

void bosluk_yazdir(int n){ // agac arasindaki bosluklari yazdirir
        int i;
        for(i = 0; i < n ; i++){
            printf(" ");
        }
}
int main(){
        system(" color F5 "); // TEMA RENK AYARI
        setlocale(LC_ALL, "Turkish");  //TURKCELESTIRME ayari
        char tus;  // metini dosyadan tarar
        char metin[MAX];
        FILE *file;
        printf(" Veri Yapýlarý Ödev-2 'Huffman Tree Coding Ödevi' \n");
        printf(" Ödevi çalýþtýrmak için huffman yazýnýz.\n Daha sonra
duygu.txt giriþini yapýnýz.\n Programdan çýkmak için ise; e tuþununa
basýp enterlayýnýz. :)\n");
        scanf("%[^\n]s",metin);

        if(tus == 'e' || tus == 'E') return -1; // eger sonlandirmak
istersek e tusa basarak programi bitirir.

        if(!strcmp(metin,"huffman")){
            dosyayi_okuma(metin , file);
        }
```

```
    return 0;
}
```

Analiz:



Programın çalışması



Programın sonlandırılması