YILDIZ TEKNİK ÜNİVERSİTESİ

ELEKTRİK-ELEKTRONİK FAKÜLTESİ

BİLGİSAYAR MÜHENDİSLİĞİ

BLM3021 – ALGORİTMA ANALİZİ

2.ÖDEV

Konu : Hashing Algoritmasının Kullanımı

Ad Soyad : Eren TUTUŞ

Okul No : 17011702

E-posta : l1117702@std.yildiz.edu.tr

Programın Çalışma Süreci

1-) void baslangicKeyDegeri ( ) :

Hash tablosunda key, kelime ve doküman adını yerleştirmeden önce her key değerini -1 değeri ile işaretledik. Böylelikle hash tablosuna değerler ekledikten sonra boş kalan indislerin hangileri olduğunu o indisin key değerini -1 ile karşılaştırıp elde etmiş olacağız.

2-) double tablodakiElemanSayisi ( ) :

Bu fonksiyonda, load factor hesaplarken bizim için gerekli olacak hash tablosundaki eleman sayısını (N) buluyoruz. Hash tablosundaki eleman sayısını bulurken 1.fonksiyonda değindiğim indislerin key değerinin -1 den farklı olması durumunda o indiste bilgiler mevcuttur ve biz ne kadar bilgi varsa onların eleman sayısını çekmiş olacağız.

3-) double calculateLoadFactor ( ) :

Load factor hesaplarken 2.fonksiyondan dönen tablo eleman sayısıda (N) formülümüze dahil olacak. Burada zaten sabit olarak tanımladığımız tablo boyutu (M) değeride formülümüzde olacak. Böylelikle Load Factor bulurken LoadFactor = Tablodaki Eleman Sayısı / Tablo Boyutu (L = N / M) işlemini yapacağız. Bu fonksiyonu ileri ki fonksiyonlarda her eleman ekleme işleminde sorguda kullanacağız.

4-) void ekleHashTable (int key, char word [ ] , char dosyaAdi [ ]) :

Hash tablosuna verileri eklerken bu fonksiyona bilgiler gönderiyoruz ve burada öncelikle key değerinin uygun indise yerleşmesi için 2 tane hashten geçirmemiz gerekiyor çünkü bu ödevimizde double hashing mantığını kullanıyoruz. İlk olarak key değeri double hashingten geçtikten sonra eğer uygun indis -1 olarak işaretliyse oraya daha önce hiç key değeri gelmemiş yani çakışma yaşanmayacak ve direk o indise key değerini, kelimeyi ve doküman adını ekleyebiliriz. Fakat gelen key değerine göre belirlenen indiste daha önce key değeri oluşmuş ise i değerini arttırıp yeniden hashing işlemi yapıyoruz bunun nedeni aynı indise değer yazmayı önlemek yani çakışmayı önlemek. Bu aşamada i değeri sürekli artarak key değeri kendisine boş bir indis bulana dek bu işleme devam edilir. Burada önemli bir diğer

husus i değeri artarken döngü içerisine bir şart koymamız gerekiyor i < TABLE_SIZE çünkü i değeri tablo boyutunu aşmamalı. Key değeri kendisine uygun bir indis bulduktan sonra bu indise key değerini, kelimeyi ve doküman adını yazacağız fakat burada bir sorgu daha yapmamız gerekiyor, bu sorgu gelen doküman adı daha önce hash tablosunda aynı indis içerisinde eklenmiş mi. Eğer aynı doküman adı varsa bir daha aynı doküman adını yazmayacak.

5-) void oncedenHashTablosuVarmi ( ) :

Bu fonksiyon aslında programın ikinci veya daha sonraki çalıştırmalarında devreye girecek bir fonksiyon bunun nedeni daha önceden oluşturulan bir hash tablosu var mı yok mu bunu sorgular. Eğer yoksa demek oluyor ki bu programa ilk defa doküman ekleyeceğiz. Ancak daha önceden bir hash tablosu oluşmuş ise o zaman yeni gelecek doküman ile birlikte önceki dökümanın hash tablosunu beraber kullanacağız. Programın ikinci veya daha sonraki çalışmalarında devreye girecek olan kısmını incelersek;

Var olan hash tablosunu dosya sonuna kadar okuma işlemi gerçekleştiriyoruz. Burada key, kelime ve doküman adını yeniden programımızda hash tablosuna alabilmemiz için key/kelime/dokumanAdi1 dokumanAdi2… şeklinde tasarladım. Böylelikle ‘ / ’ ve ‘ \ n’ karakterlerini if sorgusunda kullanıyoruz bunun sebebi key, kelime ve doküman adını birbirinden ayırıp hash tablosunda uygun şekilde yerleştirmek. Bu işlemleri yaptıktan sonra bilgileri yeniden 4.fonksiyon olan ekleHashTable ‘ a yollayarak key değerini double hashing’den geçirip gerekli indise yerleştiriyoruz.

6-) void dosyayaYaz (double loadFactor) :

Hash tablosunu dosyaya yazmak için bu fonksiyondan yararlanıyoruz. Burada key değeri -1 olmayan tüm indislerde değer olduğu için bu sorguyu yaparak hash tablosundaki key, kelime ve doküman isimlerini yazdırmış oluyoruz. Fakat buradaki başka bir önemli husus ise doküman ismi bir taneden fazla olabileceği için onuda ayrıca bir döngüde yazdırmamız gerekiyor.

7-) void dokumanOku ( ) :

Doküman okuma fonksiyonunda, dökümandan gelen kelimeleri okuyup kelimeler için horner metoduna göre key değeri oluşturup bunu 4.fonksiyon olan ekleHashTable fonksiyonuna yollayıp hash table üzerinde gerekli indise yerleştiriyoruz. Horner metodunda R = 3 aldım ve kelime için key değeri oluşturduğumuzda bu key değeri int veri tipini aşmakta bunun için unsigned long long int olarak tanımladım key değerini. Dökümanda özel karakterler olmayacağı bilgisi bize ödev dökümanında belirtildiği için kelimeler aralarında birer boşluk olarak var olduğunu varsayıyoruz. Bu nedenle dökümandan kelimeleri ayırt edebilmek için ‘ ‘ (boşluk karakteri) ve ‘ \n ‘ (satır sonu karakteri) karakterlerine göre sorgulama yapıyoruz.

Horner metodunda key = R^(L-1) * (str[0] – ‘A’ + 1) + ……. R^0 * (str[L-1] – ‘A’ +1) formülünü kullanacağız burada R = 3 , L = Kelime uzunluğu, str = Kelime. Key değeri bulunduktan sonra ekleme işlemini Load Factor değerine göre yapacak. Eğer load factor değeri 1.0 veya daha büyük ise ekleme işlemi duracak ve eklenemeyen kelimeler ekrana yazdırılacak. Eğer load factor değeri 0.8 ‘ den büyük ise 0.8 değerini geçtiğimizi bildirmek için kullanıcıya uyarı mesajı verecek ve ekleme işlemi devam edecek. Bunların dışında ise ekleme işlemi devam edecek. Hash tablosuna ekleme işlemi bittikten sonra güncel tabloyu yine dosyaya yazdırıyoruz.

8-) void kelimeAra (char word [ ]) :

Bu fonksiyon sayesinde aranan kelime hash tablosunda var mı ? eğer var ise hangi dökümanlarda yer aldığını kullanıcıya çıktı olarak verecek. Fonksiyona gelen kelimenin key adresini bilirsek hash tablosunda bulmak bizi için daha kolay olacak bu nedenle kelimeyi yeniden horner metodundan geçirip key değerini buluyoruz ve bu key değerinide tekrar double hashing’den geçiriyoruz. Böylece kullanıcının girdiği kelime eğer hash tablosunda yoksa kullanıcıya kelimenin tabloda olmadığını bildirmek için mesaj yolluyoruz eğer kelime var ise bu kelimenin geçtiği doküman isimlerini listeliyoruz.

9-) void main ( ) :

Main fonksiyonunda bir Word dizisi tanımladık bu kullanıcının arama yapacağı dizinin adı. Daha sonra hash tablosunda ilk olarak her key değerine -1 atamak için baslangicKeyDegeri ( ) fonksiyonunu çalıştırıyoruz ayrıca oncedenHashTablosuVarmi ( ) fonksiyonunu çağırarak daha önceden oluşmuş bir hash tablosu varmı onu kontrol ediyoruz. Son olarak kullanıcı arayüzünde görülen kısımda kullanıcıya doküman ekleme işlemi ve kelime aramak için tercih yapması istenir ilgili tercihe göre işlemler gerçekleştirilir.

Karmaşıklık Analizi

Kelime Arama yaparken arayacağımız kelimenin key değerini çıkarıp double hashing işlemi uyguladığımız için tek seferde hash table üzerinde bulmuş olacak bu yüzden;

Big O (1)

Eğer kelime yoksa tablodaki tüm indisleri dolaşacak bu yüzden karmaşıklık Tablo boyutuna N dersek Big O (N) olur.

Doküman ekleme olayını düşünecek olursak dökümanın içinde N tane kelime varsa o halde karmaşıklık Big O (N) olur.

C KODU

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>

#define TABLE_SIZE 997  // 1000'den küçük en büyük asal sayıyı tablo boyutu olarak aldık.

typedef struct link{
    unsigned long long int key; // Key değeri int'in alan boyunu geçtiği için unsigned long long int tanımladım.
    char word[50];  // Kelimeleri tuttuğum yer.
    char dosyaAdi[50][50];  // Dosya isimlerini tuttuğum yer.
}hashTable;

hashTable hash[TABLE_SIZE];    // Hash Tablosu
```

```
void baslangicKeyDegeri(){   // Tabloda key değerlerini ilk başta -1 olarak
işaretleme yapacağım fonksiyon.
    int i;
    for(i=0;i<TABLE_SIZE;i++){
        hash[i].key = -1;  //  Böylelikle içinde değer olmayan satırlar -1
olarak gösterilmiş oldu.
    }
}

double tablodakiElemanSayisi(){   // Load Factor hesaplarken bizim için
gerekli olan hash tablosundaki eleman sayısını(N) buluyoruz.
    int i;
    double sayac=0;
    for(i=0;i<TABLE_SIZE;i++){
        if(hash[i].key != -1){  // -1 olmayan tüm keylerde kelimeler var bu
yüzden -1 key değeri olmayanların sayısını alabiliriz.
            sayac++;
        }
    }
    return sayac;  // Tablodaki Eleman Sayısı Döner.
}

double calculateLoadFactor(){   // LoadFactor = Tablodaki Eleman Sayısı /
Tablo Boyutu ; L = N / M
    int tableSize = TABLE_SIZE;  // M -> Tablo Boyutu
    double tabloElemanSayisi = (double)tablodakiElemanSayisi();  // N ->
Tablodaki Eleman Sayısı
    double loadFactor = (double)(tabloElemanSayisi / tableSize); //
LoadFactor = N / M
    return loadFactor;
}

void ekleHashTable(int key,char word[],char dosyaAdi[]){  // Tabloya
yerleştirilecek olan bilgilerin işlendiği fonksiyon.
    int i = 0;
    int ilkHash = key % TABLE_SIZE;  // Double hashing'de iki kez hash'den
geçirmemiz gerekiyor key değerini
    int ikinciHash = 1 + (key % (TABLE_SIZE - 1) ); // İkinci hash'den
geçirdik.
    int calculatedHash = (ilkHash + i * ikinciHash) % TABLE_SIZE;  //  i =
0 değerine göre key için gerekli indis bulundu.

    if (hash[calculatedHash].key == -1){  // Eğer gelen key değeri için
belirlenen indiste -1 mevcutsa o zaman ilk defa o indise key atanacaktır.
        hash[calculatedHash].key = key;
        strcpy(hash[calculatedHash].word,word);
        strcpy(hash[calculatedHash].dosyaAdi[0],dosyaAdi);
    }
    else{// else durumunda ise hash tablosunda belirlenen indiste -1 yok
yani bu durum bize daha önce o indise bir key değeri yerleşmiş olduğunu
belirtiyor.
        while((hash[calculatedHash].key != -1) && hash[calculatedHash].key
!= key && i<TABLE_SIZE){ // i değeri tablo boyutunu açmayacak şekilde;
            i++;  //  gelen key değeri -1'den farklı mı , gelen key değeri
için daha önce aynı indiste bir key yerleşmiş mi  kontrollerini yaparız.
            calculatedHash = (ilkHash + i * ikinciHash) % TABLE_SIZE; //
böylelikle kendisine boş indis bulana dek bu döngü i sayacımız sayesinde
devam eder.
        }
        i=0;
```

```
        while(strlen(hash[calculatedHash].dosyaAdi[i]) > 0){     // Gelen
key değeri ve kelime daha önce varsa üstüne yazılır, fakat;
            i++;                                                 // farklı
dökümandan geldiyse dosya isimleri sırası ile eklenir.
        }                                                        // Bunun
için aynı dosya ismi 2.kez geldiğinde onu yazdırmamak adına
        int kontrol=0,j;                                         // bir
kontrol yapmamız gerekiyor bu nedenle önce kaç tane dosya varsa sayısını
alıyorum.
        for(j=0;j<i;j++){  // dosya sayısı kadar döngü gerçekleştiriyorum.
            if(strcmp(hash[calculatedHash].dosyaAdi[j],dosyaAdi)== 0){  //
Yeni gelen dosya daha önce eklenmiş mi onun sorgusunu yapıp bir kontrol
değeri oluşturuyorum.
                kontrol = 1;
            }
        }
        if(kontrol == 0){   // kontrol = 1 ise aynı dosya daha önce
kaydedilmiş fakat kontrol = 0 ise farklı dosya gelmiş ve kaydedebiliriz.
            hash[calculatedHash].key = key;
            strcpy(hash[calculatedHash].word,word);
            strcpy(hash[calculatedHash].dosyaAdi[i],dosyaAdi);
        }
    }
}

void oncedenHashTablosuVarmi(){  // İlk dökümanı kullandığımızda bir hash
tablosu oluşacak fakat sonradan gelen dökümanlar için önceki hash
tablosunuda kullanacağız.
    FILE *fp;
    if((fp = fopen("17011702.txt","r"))!= NULL){ // Daha önce hiç hash
tablosu oluşmuş mu bunun sorgusunu yapıyoruz.
        printf("\nDaha once olusturulan bir hash tablosu vardi simdi yeni
dokuman ismi giriniz...");
        hashTable newData;
        char temp[50];  // Sayfadaki tüm karakterleri kontrol etmek üzere
kullanacağım dizi
        char temp2[50];  // Kelime , key ve dosya adını hashtablosuna
tekrar yollarken kullanacağım geçici dizi
        char key[50]; // temp2 dizisinden gelen key değerini bu key
dizisine atacağım daha sonra bunu integer formatına dönüştürüyor olacağım.
        char word[50]; // temp2 dizisinden gelen word değerini bu word
dizisine atacağım.
        char dosyaAdi[50]; // temp2 dizisinden gelen word değerini bu
dosyaAdi dizisine atacağım.
        int sira=0;  // key , kelime ve dosya adı işlemlerinin hangisi
temp2 dizisine geldiyse kontrol sayacı olarak kullanacağım değişken.
        int i=0,j;
        int newKey;  // dosyadan okurken char tipinde olan key değerini
tekrar integer formatına dönüştüreceğiz ve o değeri bu değişkende
tutacağız.
        while(!feof(fp)){  // Dosya sonuna kadar okuma yapıyoruz.
            temp[i] = fgetc(fp);
            if(temp[i] == '/' || temp[i] == '\n'){  // Dosyada,
key/kelime/dosyaAd1 dosyaAdi2 dosyaAdi 3 şeklinde tutulduğu için '/' ve
'\n' sorgusu yapıyorum.
                if(sira == 0){  // sira == 0 ise KEY değeri gelmiştir ve
key dizisinde saklıyoruz.
                    strcpy(key,temp2);
                    sira++;
```

```
                    for(j=0;j<50;j++){  // daha sonra temp2 değişkenini
yine kullanacağım için içindeki karakterleri siliyorum.
                        temp2[j] = '\0';
                    }
                    i=0;
                }
                else if(sira == 1){   // sira == 1 ise KELİME gelmiştir ve
word dizisinde saklıyoruz.
                    strcpy(word,temp2);
                    sira++;
                    for(j=0;j<50;j++){ // daha sonra temp2 değişkenini yine
kullanacağım için içindeki karakterleri siliyorum.
                        temp2[j] = '\0';
                    }
                    i=0;
                }
                else{      // sira = 3 olmuştur ve son işlem olan dosya
isimlerini aktarma ve daha önce word ve key dizisinde tuttuklarımızıda
tekrar tabloya aktaracağız.
                    strcpy(dosyaAdi,temp2); // temp2 deki dosya adını
dosyaAdi dizisine aktardık.
                    newKey = atoi(key); // key değerinde char olarak
tutulan değeri integer olarak düzenledik.
                    ekleHashTable(newKey,word,dosyaAdi);  // ekleHashTable
fonksiyonuna key değerini belirleyip hash tablosuna aktarmak üzere
yolladık.
                    sira = 0;
                    for(j=0;j<50;j++){
                        temp2[j] = '\0';  // her seferinde yeni değerler
için kullanacağımız geçici temp2 değişkeninin içindekileri siliyoruz.
                    }
                    for(j=0;j<50;j++){
                        key[j] = '\0';   // her seferinde yeni değerler
için kullanacağımız geçici key değişkeninin içindekileri siliyoruz.
                        word[j] = '\0';   // her seferinde yeni değerler
için kullanacağımız geçici word değişkeninin içindekileri siliyoruz.
                        dosyaAdi[j] = '\0';   // her seferinde yeni
değerler için kullanacağımız geçici dosyaAdi değişkeninin içindekileri
siliyoruz.
                    }
                    i=0;
                    newKey = 0;
                }
            }
            else{
                temp2[i] = temp[i];  // temp ile son işlemde her zaman özel
karakter geleceği için key,kelime ve dosya adını temp2'de tutuyorum.
                i++;
            }
        }
    }
}

void dosyayaYaz(double loadFactor){   // Hash Tablosunu dosyaya yazdığımız
fonksiyon
    FILE *fp;
    if((fp = fopen("17011702.txt","w+")) == NULL){
        printf("Dosya acma hatasi");
    }

    int i,j=0;
```

```
    for(i=0;i<TABLE_SIZE;i++){
        if(hash[i].key != -1){      // -1 değerine eşit olan key'lere bilgi
gelmemiştir ve onları haricinde olan tüm bilgileri döndürebiliriz.
            fprintf(fp,"%d/%s/",hash[i].key,hash[i].word);  // her gelen
satır için key, word değerini yazdırırız.
            j=0;
            while(strlen(hash[i].dosyaAdi[j]) > 0){  // dosya adında ise 1
den fazla dosya adı ismi olabileceği için onuda kendi içerisinde döngüye
sokuyoruz.
                fprintf(fp,"%s",hash[i].dosyaAdi[j]);
                j++;
            }
            fprintf(fp,"%c\n",' ');
        }
    }
    fprintf(fp," Load Factor  = %.1f",loadFactor);
    fclose(fp);
}

void dokumanOku(){  // Dökümandan gelen kelimeleri okuyup kelimeler için
horner metoduna göre key değeri oluşturup bunu ekleHashTable fonksiyonuna
yollayıp hashtable üzerinde gerekli indise yerleştiriyoruz.
    char dosyaAdi[50];
    printf("\nAcilacak dosya adi giriniz ( Ornek : dosyaAdi.txt ) : ");
    scanf("%s",dosyaAdi);
    FILE *fp;
    if((fp = fopen(dosyaAdi,"r"))== NULL){   // Dosya okuma kontrolü
        printf("Dosya acma hatasi");
    }
    int i=0,j;
    double loadFactor; // loadFactor değişkeni
    unsigned long long int key=0;  // key değeri int değerini aşacağı için
unsigned long long int yaptım.
    char temp[50];   // Dökümandaki tüm karakterleri kontrol olarak
kullanmak için alacağım geçici dizi
    char temp2[50];  // Dökümandaki tüm kelimeleri alacağım geçici dizi
    while(!feof(fp)){
        temp[i] = fgetc(fp);
        if(temp[i] == ' ' || temp[i] == '\n'){   // Dökümanda kelimeler
arasında boşluk (' ') olacağı için kontrol yapıyorum her boşlukta kelimeyi
alıp hash tablosuna aktarıyorum.
            int length = strlen(temp2);          // Ayrıca satır sonuna
gelince ('\n') satırdaki son kelimeyide alıp alt satır için işleme devam
eder.
            int power = length-1;  //  Horner metodunda R^(Length-1)
alacağı için üs değerini belirliyoruz ve döngüde 0 a kadar azalacak şekilde
işleme sokuyoruz.
            key = 0;
            char gecici;
            for(j=0;j<length;j++){
                gecici = temp2[j];
                if (gecici >= 'a'){     // Araba ve araba aynı kelimeler
gibi düşüneceğiz bu nedenle kontrolü gerçekleştirdik.
                    gecici = 'A' - 'a' + gecici;
                }
                key += abs((pow(3,power)*(gecici-'A'+1)));   // Horner
metodunda gelen kelimenin karakter sayısını kullanarak döngüye sokuyoruz
işlem bittiğinde kelime için key değeri oluşacaktır.
                power--;
            }
```

```
            loadFactor = calculateLoadFactor(); // calculateLoadFactor
fonksiyonu ile Load Factor sayısını çektik.
            printf("LoadFactor : %.1f\n",loadFactor);
            if(loadFactor == 1.0 || loadFactor > 1.0){  // Eğer LoadFactor
>= 1.0 ise Ekleme İşlemi Duracak ve Eklenemeyen Kelimeler Ekrana Yazılacak.
                printf("Eklenemeyen Kelime : %s\n",temp2);
                for(j=0;j<50;j++){ // temp2 daha sonraki kelimelerde
kullanacağımız için içini boşalttık.
                    temp2[j] = '\0';
                }
                i=0;
            }
            else{
                if(loadFactor > 0.8){   // Load Factor > 0.8 'den 0.8
değerini geçtiğimizi bildirmek için kullanıcıya uyarı mesajı verecek.
                    printf("LOAD FACTOR 0.8 degerini gecti\n");
                }

                if (key != 0){
                    ekleHashTable(key,temp2,dosyaAdi);  // Gelen key
değeri,kelime ve dosyaAdina göre Hash tablosuna eklenmek üzere
ekleHashTable fonksiyonuna yollanıyor.
                }

                for(j=0;j<50;j++){  // Gecici temp2 degiskeninin içini
boşaltıyoruz.
                    temp2[j] = '\0';
                }

                i=0;
            }
        }
        else{
            temp2[i] = temp[i];  // temp ile son işlemde her zaman özel
karakter geleceği için kelimeyi temp2'de tutuyorum.
            i++;
        }
    }
    dosyayaYaz(loadFactor);  // Daha sonra dosyaya yazdırıyorum oluşan hash
tablosunu ve loadfactor değerinide yolluyorum.
    fclose(fp);
}

void kelimeAra(char word[]){  // Bu fonksiyon sayesinde aranan kelime hash
tablosunda var mı ? eğer var ise hangi dökümanlarda yer aldığını
kullanıcıya çıktı olarak verecek.
    int i=0,adimSayisi=1,j;
    unsigned long int key=0;  // key değeri int değerini aşacağı için
unsigned long long int yaptım.
    int length = strlen(word);
    int power = length-1; //  Horner metodunda R^(Length-1) alacağı için üs
değerini belirliyoruz ve döngüde 0 a kadar azalacak şekilde işleme
sokuyoruz.
    char gecici;
    for(j=0;j<length;j++){
        gecici = word[j];
        if (gecici >= 'a'){  // Araba ve araba aynı kelimeler gibi
düşüneceğiz bu nedenle kontrolü gerçekleştirdik.
            gecici = 'A' - 'a' + gecici;
        }
```

```
        key += abs((pow(3,power)*(gecici-'A'+1)));  // Horner metodunda
gelen kelimenin karakter sayısını kullanarak döngüye sokuyoruz işlem
bittiğinde kelime için key değeri oluşacaktır.
        power--;
    }
    int ilkHash = key % TABLE_SIZE;  // Double hashing'de iki kere hash'den
geçirip indis değeri üreteceğimiz için
    int ikinciHash = 1 + (key % (TABLE_SIZE - 1) ); // İkinci hash işlemi
    int calculatedHash = (ilkHash + i * ikinciHash) % TABLE_SIZE; // Ve
indis değeri oluştu.
    if(hash[calculatedHash].key == key){   // İlgili indiste bulunan key
değeri ile kullanıcının göndermiş olduğu kelimenin key değeri birbirine
eşit mi ?
        printf("\n%d adim gerceklesti.",adimSayisi);
        i=0;

        while(strlen(hash[calculatedHash].dosyaAdi[i]) > 0){     // kaç
tane döküman ismi varsa sayısını tutuyoruz.
            i++;
        }
        int dosyaSayisi = i;
        for(i=0;i<dosyaSayisi;i++){  // döküman sayısı kadar döngü kurup
kelimenin yanına dosya isimlerini tek tek yazıdırıyoruz.
        printf("\n%s -->
%s",hash[calculatedHash].word,hash[calculatedHash].dosyaAdi[i]);
        }
    }
    else{
        printf("\n%d adim gerceklesti ve aranan kelime tabloda mevcut
degil.",adimSayisi);
    }
}


int main(){
    char word[50];   // Aranacak kelimenin tutulduğu dizi
    baslangicKeyDegeri();   // Hash tablosunda başlangıçta tüm KEY
değerlerini -1 olarak işaretliyoruz.
    oncedenHashTablosuVarmi();   // Daha önceden hash tablosu varmı onu
kontrol ediyoruz.
    printf("\n Dokuman Eklemek icin '1' tusuna basin  /  Kelime Aramak icin
'2' tusuna basin");
    int secim;
    scanf("%d",&secim);

    if(secim == 1){
        dokumanOku();   // // Dökümandan gelen kelimeleri okuyup kelimeler
için horner metoduna göre key değeri oluşturup bunu ekleHashTable
fonksiyonuna yollayıp hashtable üzerinde gerekli indise yerleştirmek üzere
çağırdığımız fonksiyon
    }
    else if(secim == 2){
        printf("\nArayacaginiz kelimeyi giriniz : ");
        scanf("%s",word);
        kelimeAra(word);  // Kelime arama işlemi
    }
    else{
        printf("Yanlis Tercih Yaptiniz !");
    }
    return 0;
}
```

EKRAN ÇIKTILARI

C:\Users\erent\Desktop\SON\17011702.exe

```
 Dokuman Eklemek icin '1' tusuna basin  /  Kelime Aramak icin '2' tusuna basin1

Acilacak dosya adi giriniz ( Ornek : dosyaAdi.txt ) : words.txt
LoadFactor : 0.0
LoadFactor : 0.1
LoadFactor : 0.1
LoadFactor : 0.2
LoadFactor : 0.3
LoadFactor : 0.3
LoadFactor : 0.4
LoadFactor : 0.4
LoadFactor : 0.5
LoadFactor : 0.6
LoadFactor : 0.6
LoadFactor : 0.6
LoadFactor : 0.7

--------------------------------
Process exited after 4.456 seconds with return value 0
Press any key to continue . . .
```

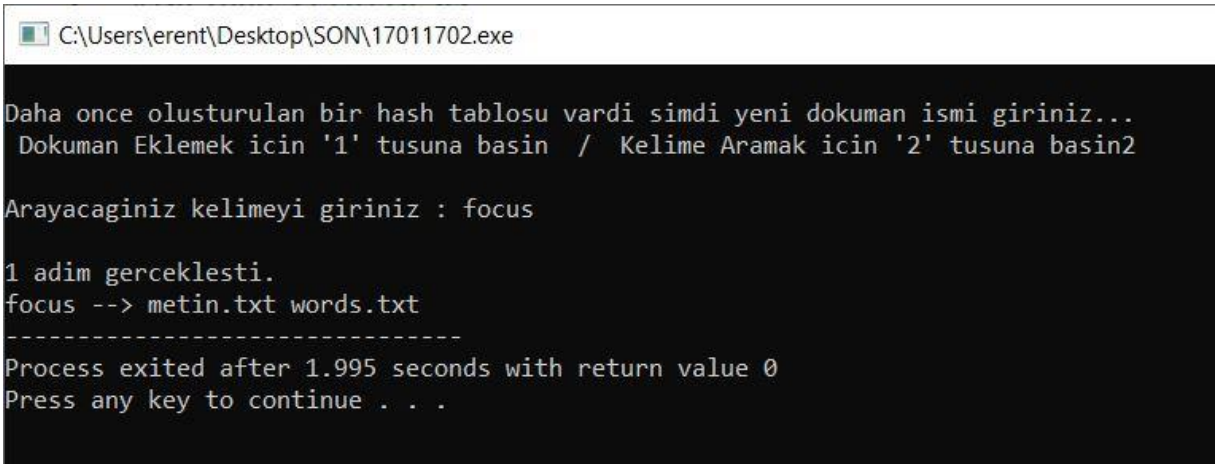
```
LoadFactor : 0.5
LoadFactor : 0.5
LoadFactor : 0.5
LoadFactor : 0.5
LoadFactor : 0.5
LoadFactor : 0.5
LoadFactor : 0.5
LoadFactor : 0.5
LoadFactor : 0.5
LoadFactor : 0.5
LoadFactor : 0.5
LoadFactor : 0.5
LoadFactor : 0.5
LoadFactor : 0.5
LoadFactor : 0.5
LoadFactor : 0.5
LoadFactor : 0.5
LoadFactor : 0.5
LoadFactor : 0.5
LoadFactor : 0.5
LoadFactor : 0.5
LoadFactor : 0.5
LoadFactor : 0.5
LoadFactor : 0.5

--------------------------------
Process exited after 2.653 seconds with return value 0
Press any key to continue . . .
```

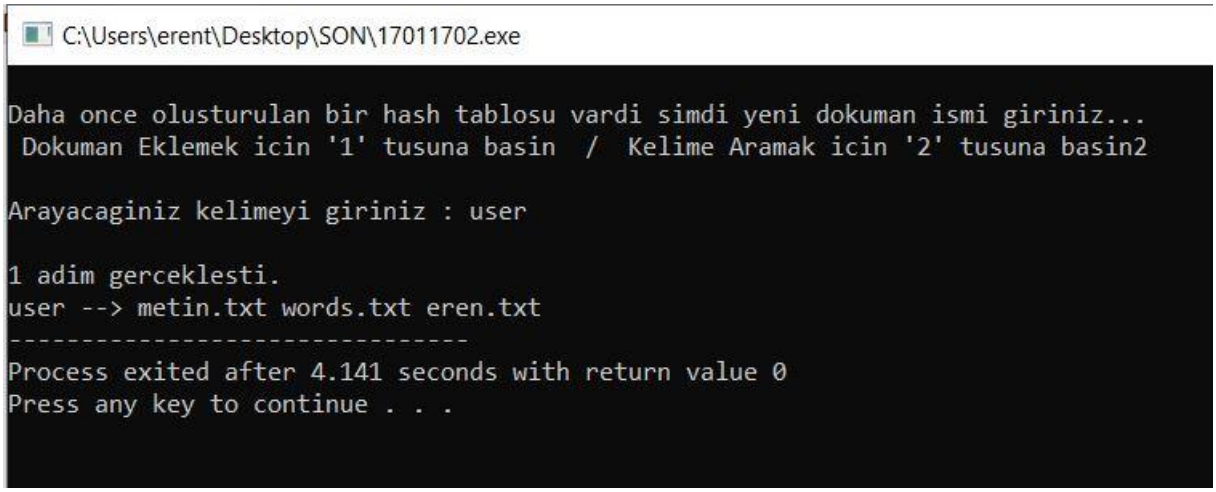
C:\Users\erent\Desktop\SON\17011702.exe

```
LoadFactor : 0.9
LOAD FACTOR 0.8 degerini gecti
LoadFactor : 0.9
LOAD FACTOR 0.8 degerini gecti
LoadFactor : 0.9
LOAD FACTOR 0.8 degerini gecti
LoadFactor : 0.9
LOAD FACTOR 0.8 degerini gecti
LoadFactor : 0.9
LOAD FACTOR 0.8 degerini gecti
LoadFactor : 0.9
LOAD FACTOR 0.8 degerini gecti
LoadFactor : 0.9
LOAD FACTOR 0.8 degerini gecti
LoadFactor : 0.9
LOAD FACTOR 0.8 degerini gecti
LoadFactor : 0.9
LOAD FACTOR 0.8 degerini gecti
LoadFactor : 0.9
LOAD FACTOR 0.8 degerini gecti

--------------------------------
Process exited after 3.108 seconds with return value 0
Press any key to continue . . .
```

C:\Users\erent\Desktop\SON\17011702.exe

```
LoadFactor : 1.0
Eklenemeyen Kelime : enterprise
LoadFactor : 1.0
Eklenemeyen Kelime : capability
LoadFactor : 1.0
Eklenemeyen Kelime : innovation
LoadFactor : 1.0
Eklenemeyen Kelime : stakeholder
LoadFactor : 1.0
Eklenemeyen Kelime : core
LoadFactor : 1.0
Eklenemeyen Kelime : competency
LoadFactor : 1.0
Eklenemeyen Kelime : architecture
LoadFactor : 1.0
Eklenemeyen Kelime : team
LoadFactor : 1.0
Eklenemeyen Kelime : player
LoadFactor : 1.0
Eklenemeyen Kelime : milestone
LoadFactor : 1.0
Eklenemeyen Kelime : executive
LoadFactor : 1.0
Eklenemeyen Kelime : search
LoadFactor : 1.0
Eklenemeyen Kelime : core

--------------------------------
Process exited after 10.31 seconds with return value 0
Press any key to continue . . .
```


C:\Users\erent\Desktop\SON\17011702.exe
Daha once olusturulan bir hash tablosu vardi simdi yeni dokuman ismi giriniz...
Dokuman Eklemek icin '1' tusuna basin / Kelime Aramak icin '2' tusuna basin2
Arayacaginiz kelimeyi giriniz : focus
1 adim gerceklesti.
focus --> metin.txt words.txt
Process exited after 1.995 seconds with return value 0
Press any key to continue . . .


C:\Users\erent\Desktop\SON\17011702.exe
Daha once olusturulan bir hash tablosu vardi simdi yeni dokuman ismi giriniz...
Dokuman Eklemek icin '1' tusuna basin / Kelime Aramak icin '2' tusuna basin2
Arayacaginiz kelimeyi giriniz : user
1 adim gerceklesti.
user --> metin.txt words.txt eren.txt
Process exited after 4.141 seconds with return value 0
Press any key to continue . . .