ALGORİTMA ANALİZİ

ÖDEV 2 - GRUP 1

İrem ATILGAN

17061036

12.02.2020

FONKSİYONLAR

1. search(char*, HashTable*) : Hash tablosunda kelime arama fonksiyonudur. Parametre olarak aratılan kelime ve mevcut hash tablosu alınır. Kelimenin geçtiği dökümanlar varsa isimleri ve kaç adımda bulunduğu yazdırılır.

2. insertTable(char*,char*,HashTable*) : Hash tablosuna döküman ve döküman kelimelerini yerleştiren fonksiyon. Yerleştirmeler tamamlandıktan sonra oluşan tablo döndürülür. Parametre olarak okunan doküman bloğu, dökümanın adı ve mevcut hash tablosu alınır.

3. createTable() : HashTable struct'ını başlatan fonksiyondur. Hash tablosu sonuç olarak dönülür.

4. readDocument(char*) : Döküman dosyasını okuma fonksiyonudur. Parametre olarak dökümanın ismi alınır. Dökümandaki kelimeler char dizisi olarak dönülür.

5. readTable(FILE*,HashTable*) : "hashTable.txt" dosyasındaki verileri hash tablosuna (HashTable struct'ına) yerleştiren fonksiyon. Parametre olarak dosya (FILE pointer) ve başta oluşturulan hash tablosu alınır; Okunduktan sonra oluşan hash tablosu dönülür.

6.long int hornersMethod(char*,int) : Verili kelimeden anahtar değer hesaplayan Horner's Yöntemi'dir. Parametre olarak kelime ve kelimenin uzunluğu alınır; Metot sonucu oluşan anahtar değer dönülür.

7. updateFile(FILE*,HashTable*) : Hash tablosunu dosyaya yazan fonksiyondur. Parametre olarak dosya (FILE pointer) ve hash tablosu alınır.

8. createDocs() : CELL struct'ında tutulan dökümanları initialize eden fonksiyondur. Dökümanların isimleri için yer ayrılır ve ayrılan yer char** olarak dönülür.

Aşağıdaki fonksiyonlar hash tablosunda çakışma (collision) yaşanmaması için uygulanan double hashing yönteminde yer alan hashing fonksiyonlarıdır :

9. hash1(int)

10. hash2(int)

ANALİZ

Arama Fonksiyonu : Kelime hash tablosunda aranırken ilk olarak kelimenin anahtar değeri Horner's Metodu ile hesaplanır ve double hashing yoluyla bulunan adresteki hücreye hash tablosunda bakılır. Eğer hücredeki anahtar değer ile aradığımız kelimenin anahtar değeri uyuşmazsa tekrar hash değeri hesaplanır. Aranan kelime bulunursa o hücredeki dökümanlar sırasıyla gezilir ve konsolda yazdırılır.

Doküman sayısını K, tablo boyutunu M olarak alırsak,

Best Case : Hücre ilk hesaplanan adreste bulunur. Doküman sayısı kadar iterasyon yapılır (doküman isimlerinin yazdırılması için). Bu durumda karmaşıklık O(K) olur.

Worst Case : Tüm tablo gezilerek kontrol yapılır (M) ve doküman sayısı kadar iterasyon yapılır. Bu durumda karmaşıklık O(M*K) olur.

PROGRAM KODLARI

Structlar ve Fonksiyonların Deklarasyonu

```c
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <string.h>
#include <time.h>
#define esc 27
#define TABLE_SIZE 997

//HashTable struct'ında yer alan hücreler ayrı bir structta tutulur
typedef struct CELL{
    char** docs;         //Hücredeki kelimeyi içeren dökümanların isimleri yer alır
    long int hashedWord; //Horner's metodu ile elde edilen anahtar değeri tutar
    short document_cursor; //Hücredeki döküman sayısını tutan değişken
}CELL;

//Hash tablosu struct'ı
typedef struct HashTable{
    int M;  //Tablo boyutu
    int N;  //Tablodaki kelime sayısı
    CELL* cells; //Tabloda kelimeleri tutan hücreler

}HashTable;


void search(char*, HashTable*);                 //Hash tablosunda kelime arama fonksiyonu
HashTable* insertTable(char*,char*,HashTable*); //Hash tablosuna döküman ve döküman kelimelerini yerleştiren fonksiyon
HashTable* createTable();                       //HashTable struct'ını başlatan (initializing) fonksiyon
char *readDocument(char*);                      //Döküman dosyasını okuma fonksiyonu
HashTable* readTable(FILE*,HashTable*);         //hashTable.txt dosyasındaki verileri hash tablosuna yerleştiren fonksiyon
long int hornersMethod(char*,int);              //Verili kelimeden anahtar değer hesaplayan Horner's Yöntemi
void updateFile(FILE*,HashTable*);              //Hash tablosunu dosyaya yazan fonksiyon
char** createDocs();                            //CELL struct'ında tutulan dökümanları initialize eden fonksiyon
int hash1(int);
int hash2(int);

int main()
{
    char flag = 1;  //Programın istenmediği sürece açık kalmasını sağlayan değişken
    char ch;        //İstenen işlem türü için kullanıcıdan alınan input'u tutan değişken
    char docName[50], fileName[50], word[20]; //Döküman adı, dosya adı (.txt uzantılı), kullanıcını aradığı kelime
    HashTable* hashTable;
    HashTable* finalTable; //insertion işleminden sonra oluşan yeni hash tablosu
    FILE* fp;
```

Main Fonksiyonu

```c
int main()
{
    char flag = 1;  //Programın istenmediği sürece açık kalmasını sağlayan değişken
    char ch;        //İstenen işlem türü için kullanıcıdan alınan input'u tutan değişken
    char docName[50], fileName[50], word[20]; //Döküman adı, dosya adı (.txt uzantılı), kullanıcını aradığı kelime
    HashTable* hashTable;
    HashTable* finalTable; //insertion işleminden sonra oluşan yeni hash tablosu
    FILE* fp;

    do {
        printf("\n\t=== MAIN MENU ===");
        printf("\n\tOptions : \n");
        printf("\t1. If you want to insert a new word please type 0\n");
        printf("\t2. If you want to search for a word please type 1\n");
        printf("\n\tPress ESC to quit\n");
        ch = getch(); //Kullanıcı input'u alınır
        hashTable = createTable(); //Tablo başlatılır
        switch(ch)
        {
            //ESC tuşuna basılırsa program sonlanır
            case esc:
                flag = 0;
                break;
            case '0':
                //Tabloya değer yerleştirilmesi istenirse
                printf("\n\tCase 1\n");
                printf("\tInsert document name here : ");
                scanf("%s",&docName);
                strcpy(fileName,docName);
                strcat(fileName,".txt");
                printf("\n\tDocument Name = %s.txt",docName);
                char* doc = (char*)readDocument(fileName);
                //Döküman dosyası mevcut mu kontrol ediir
                if(doc != NULL)
                {
                    finalTable = insertTable(doc,docName,hashTable); //Dökümandaki kelimeler tabloya yerleştirilir
                    fp = fopen("hashTable.txt","w");
                    updateFile(fp,hashTable);                        //Son tablo dosya olarak yazdırılır
                    fclose(fp);
                }
                else
                    printf("\n\tFile does not exist!..");
                break;
            case '1':
                //Dökümanlarda kelime aramak istenirse
                printf("\tCase 2\n");
                printf("\tInsert word here : ");
                scanf("%s",&word);
                search(word,hashTable); //Kelime hash tablosunda aratılır
                break;
            default:
                printf("\n\tPlease type valid commands!..\n");
                break;
        }
    }
    while(flag);

    return 0;
}
```

readDocument Fonksiyonu

```
96  //Döküman dosyasını okuma fonksiyonu
97  char *readDocument(char* docName)
98  {
99      char text[10000];
100     FILE* fp;
101     if(fp = fopen(docName,"r")) //Döküman mevcutsa
102     {
103         fread(text,sizeof(char)*(500*20),1,fp); //Dökümandaki tüm kelimeler okunur
104         fclose(fp);
105
106         return text;
107     }
108
109     return NULL; //Döküman mevcut değilse NULL dönülür
110 }
```

createTable ve createDocs Fonksiyonu

```
170 //HashTable struct'ını başlatan (initializing) fonksiyon
171 HashTable* createTable()
172 {
173     HashTable* ht = (HashTable*)malloc(sizeof(HashTable));
174     ht->M = TABLE_SIZE; //Tablo boyutu atanır
175     ht->N = 0;          //Tablodaki kelime sayısı sıfırlanır
176     ht->cells = (CELL*)malloc(sizeof(CELL)*TABLE_SIZE); //Tablo hücreleri için yer ayrılır
177
178     int i;
179     for(i = 0; i < 997; i++) {
180         //Tablodaki her bir hücre henüz değer yerleştirilmediğini belirtmek üzere -1 ile işaretlenir
181         ht->cells[i].hashedWord = -1;
182     }
183
184
185     return ht;
186
187 }
188
189 //CELL struct'ında tutulan dökümanları initialize eden fonksiyon
190 char** createDocs()
191 {
192     int i;
193     //Her hücrede 50 dökümanlık yer oluşturulur ve her bir döküman ismi en fazla
194     //20 karakter olacak şekilde ayarlanır
195     char** docs = (char**)malloc(sizeof(char*)*50);
196     for(i = 0; i < 50; i++)
197         docs[i] = (char*)malloc(sizeof(char)*20);
198
199     return docs;
200 }
```

readTable Fonksiyonu

```
112 //hashTable.txt dosyasındaki verileri hash tablosuna yerleştiren fonksiyon
113 HashTable* readTable(FILE* fp, HashTable* hTable)
114 {
115     float loadF; //load factor
116     int adr;    //hash tablosu adresi değişkeni
117     long int key; //Horner's Metodu ile hesaplayacağımız anahtar değerini tutan değişken
118     char line[200]; //Okunan satırları tutan char dizisi
119     char *word; //strtok ve strtol ara işlemleri için kullanılan geçici char* değişkeni
120     char newstr[15]; //strtok ve strtol işlemleri sonucu word'ün adresinde yer alan kelimenin kopyalandığı char dizisi
121     char *tmp; //strtol işlemi için geçici oluşturulan char pointer
122     char* newline; //satırlardaki newline karakterlerin adresini tutan char pointer
123 
124     printf("\n\tREADING TABLE!..");
125     fscanf(fp,"%lf",&loadF); //Tablo dosyasındaki ilk satır olan load factor değeri okunur
126     fscanf(fp,"%d %d",&hTable->N,&hTable->M); //İkinci satırda tablodaki eleman sayısı ve tablo boyutu okunur
127     fscanf(fp,"\n");
128     fgets(line,200,fp); //Buradan itibaren tablodaki her satır dosya sonuna kadar sırayla okunur
129     while(!feof(fp))
130     {
131         //newline ('\n') karakter içeren adres bulunur ve '\0' olarak güncellenir
132         newline = strchr(line,'\n');
133         if(newline)
134           *newline = '\0';
135 
136         //Empty space karakteri ile elde edilen satır ayrıştırılır, adres ve anahtar değişkenlerine atanır
137         word = strtok(line," ");
138         strcpy(newstr,word);
139         adr = strtol(word,&tmp,10);
140         word = strtok(NULL," ");
141         strcpy(newstr,word);
142         key = strtol(word,&tmp,10);
143 
144         //adreste kaç döküman olduğunu tutan document_cursor değişkeni sıfırlanır
145         hTable->cells[adr].document_cursor = 0;
146 
147         //Hash Tablosunun verilen adresindeki dökümanlar için yer açılır
148         hTable->cells[adr].docs = createDocs();
149 
150         word = strtok(NULL," ");
151         while(word != NULL) //Empty space karakteri ile ayrılacak kelime kalmayana dek ayrıştır
152         {
153             strcpy(newstr,word);
154             strcpy(hTable->cells[adr].docs[hTable->cells[adr].document_cursor],newstr); //Hash tablosuna döküman adı yerleştirilir
155             (hTable->cells[adr].document_cursor)++; //Döküman sayısı artar
156             word = strtok(NULL," ");
157         }
158 
159         //kelimenin Horner's Method ile oluşturulmuş anahtarı yerleştir
160         hTable->cells[adr].hashedWord = key;
161 
162         //Gelecek satırı oku
163         fgets(line,200,fp);
164     }
165     printf("\n\tREADING IS DONE!..");
166     return hTable;
167 }
```

search Fonksiyonu

```c
//Hash tablosunda kelime arama fonksiyonu
void search(char* word, HashTable* hashTable)
{
    FILE* fp;
    HashTable* hTable;

    //Hash tablosu dosyasının varlığı kontrol edilir
    if(fp = fopen("hashTable.txt","r")) {
        char** tmp;     //Geçici olarak döküman adlarını tutan string dizisi
        short cursor;   //Döküman string'i içerisinde ilerlerken kullanacağımız indis değişkeni
        int k = 0;      //Collision olma durumunda yeni adresi bulurken düzenli olarak arttıracağımız değişken
        hTable = readTable(fp,hashTable); //Belgede yer alan hash tablosu HashTable struct'ına aktarılır

        //Horner's Yöntemi ile kelimenin anahtarı oluşturulur
        long int key = hornersMethod(word,strlen(word));
        //Double hashing ile yeni adres hesaplanır
        int hash = (hash1(key) + k*hash2(key))%(hashTable->M);

        if(hashTable->cells[hash].hashedWord != -1) //Adres boş değilse
        {
            //Tüm tablo dolaşılmadığı ve bulduğumuz adresteki kelime ile aradığımız kelime eşleşmediği sürece
            //k değişkenini arttırmaya ve yeni adres değerini hesaplamaya devam et
            while(k < hashTable->M && hashTable->cells[hash].hashedWord != key)
            {
                k++;
                hash = (hash1(key) + k*hash2(key))%(hashTable->M);
            }

            if(k < hashTable->M) //Kelime bulunduysa
            {
                printf("\n\tWORD FOUND!..TOTAL STEPS = %d",k+1); //Kelimenin kaç adım sonra bulunduğunu yazdır
                printf("\n\tIN DOCUMENTS : ");
                tmp = hashTable->cells[hash].docs;
                for(cursor = 0; cursor < hashTable->cells[hash].document_cursor; cursor++)
                    printf("%s\t",tmp[cursor]);

                printf("\n");
            }
            else
            {
                printf("\n\tThis word does not exist in the document!..");
            }


        }
        else //Adreste bir kelime yer almıyorsa kelime tabloda yok demektir
        {
            printf("\n\tThis word does not exist in the document!..");
        }
        fclose(fp);
    }
    else //Tablo dosyası bulunamadıysa
    {
        printf("\n\tHASH TABLE NOT FOUND!..");
    }
}
```

hash1 ve hash2 Fonksiyonu

```
259  int hash1(int key)
260  {
261      return key%(TABLE_SIZE);
262  }
263
264  int hash2(int key)
265  {
266      return (1+(key%(TABLE_SIZE-1)));
267  }
```

hornersMethod Fonksiyonu

```
420  //Verili kelimeden anahtar değer hesaplayan Horner's Yöntemi
421  long int hornersMethod(char* word, int length)
422  {
423      long int key = 0;
424      int i;
425      int r = 31; //asal sayı
426
427      i = length;
428      while(i >= 0)
429      {
430          key += word[i]*pow(r,i);
431          key = key & 0x7FFFFFFF; //Overflow olmasını engellemek için
432          i--;
433      }
434
435      return key;
436  }
```

insertTable Fonksiyonu

```
269 //Hash tablosuna döküman ve döküman kelimelerini yerleştiren fonksiyon
270 HashTable* insertTable(char* document, char* documentName, HashTable* hashTable)
271 {
272     clock_t begin = clock();
273     FILE* fp;
274     HashTable* hTable;
275     int i = 0;
276     int j = 0;
277     int k = 0;          //Collision olma durumunda yeni adresi bulurken düzenli olarak arttıracağımız değişken
278     int cursor;         //Döküman string dizisi içerisinde ilerlerken kullanacağımız indis değişkeni
279     long int key;       //Horner's Metodu ile hesaplayacağımız anahtar değerini tutan değişken
280     char word[50];      //Okunacak dökümandaki kelimeleri tutan char dizisi
281     int hash;
282     float loadF = 0;
283
284     //Tablo dosyası mevcutsa dosya okunarak HashTable struct'ına aktarılır
285     if(fp = fopen("hashTable.txt","r")) {
286         hTable = readTable(fp,hashTable);
287         fclose(fp);
288     }
```

```
290     //Dökümanın sonuna kadar harfleri oku
291     while(document[i] != '\0')
292     {
293         //Her boşluk veya newline'da kelimeyi al
294         if(document[i] == ' ' || document[i] == '\n'){
295
296             word[j] = '\0';
297             key = hornersMethod(word,j); //Kelimeden anahtar çıkarılır
298             hash = (hash1(key) + k*hash2(key))%(hashTable->M); //Double hashing ile adres hesaplanır
299             k+=1;
300
301             //load factor 1 olmadığı sürece tablo üzerinde güncellemeler yapılır
302             if(loadF < 1)
303             {
304                 //Tüm tablo dolaşılmadığı, bulunan adres boş olmadığı ve bulduğumuz adresteki kelime ile aradığımız kelime eşleşmediği sürece
305                 //k değişkenini arttırmaya ve yeni adres değerini hesaplamaya devam et
306                 while(k < hashTable->M && hashTable->cells[hash].hashedWord != -1 && key != hashTable->cells[hash].hashedWord) //Collision'dan kaçınmak için
307                 {
308                     hash = (hash1(key) + k*hash2(key))%(hashTable->M);
309                     k+=1;
310                 }
311
312                 if(k < hashTable->M) //Boş yer ya da aynı kelime bulunmuşsa
313                 {
314
315                     if(hashTable->cells[hash].hashedWord == -1) //Boş yer bulunduysa
316                     {
317                         //Boş hücreye yerleştirilecek kelimenin anahtar değeri, ve kelimeyi içeren dökümanın adı yerleştirilir
318                         hashTable->cells[hash].hashedWord = key;
319                         hashTable->cells[hash].document_cursor = 1; //Hücredeki döküman sayısı güncellenir
320                         hashTable->cells[hash].docs = createDocs(); //Hücreye döküman isimlerinin yerleştirilmesi için yer açılır
321                         strcpy(hashTable->cells[hash].docs[0],documentName); //Döküman ismi ilk döküman olarak yerleştirilir
322
323                         (hashTable->N)++; //Tabloda yer alan kelime sayısı arttırılır
324                         loadF = (float)(hashTable->N)/(hashTable->M); //Load Factor hesaplanır
325
326                         printf("\n\t");
327                         if(loadF > 0.8) //Load Factor 0.8'den büyükse uyarı ver
328                             printf("WARNING..");
329
330                         printf("LOAD FACTOR = %lf",loadF);
331                         if(loadF >= 1) //Load Factor 1 olursa dökümanda bundan sonraki kelimelerin eklenemeyeceğini haber ver
332                         {
333                             printf("\n\tTABLE IS FULL!..");
334                             printf("\n\tDISCARDED WORDS = ");
335                         }
336
337                     }
```

```
                else //Adreste yer alan aynı kelime dökümandaki kelimeyle eşleşiyorsa
                {

                    cursor = 0;
                    //Döküman isminin hücrede yer alan dökümanlar arasında olup olmadığını kontrol et
                    while(cursor < hashTable->cells[hash].document_cursor && (strcmp(documentName,hashTable->cells[hash].docs[cursor])))
                    {
                        cursor++;
                    }

                    if(cursor >= hashTable->cells[hash].document_cursor) //Döküman hücrede bulunmadıysa ismini döküman dizisine ekle
                    {
                        strcpy(hashTable->cells[hash].docs[hashTable->cells[hash].document_cursor],documentName);
                        (hashTable->cells[hash].document_cursor)++; //Döküman sayısının kaydını arttır
                    }


                }
            }
            else
            {
                printf("\n\tThere is no available space for word %s in hash table",word);
            }
        }
        else //Load factor 1 ise eklenemeyen kelimeler yazdırılır
        {
            printf("\n\t%s",word);
        }


        k = 0;
        j = 0;
        i++;

    }
    if(document[i] <= 90) //Upper case harf varsa lower case'e çevrilerek aktarılır
        word[j++] = document[i++] - ('A'-'a');
    else
        word[j++] = document[i++];

}

clock_t end = clock();
double time_spent = (double)(end-begin) / CLOCKS_PER_SEC; //fonksiyonun çalışma süresini yazdır
printf("\n\tINSERTION TIME = %lf\n",time_spent);



return hashTable;
}
```

updateFile Fonksiyonu

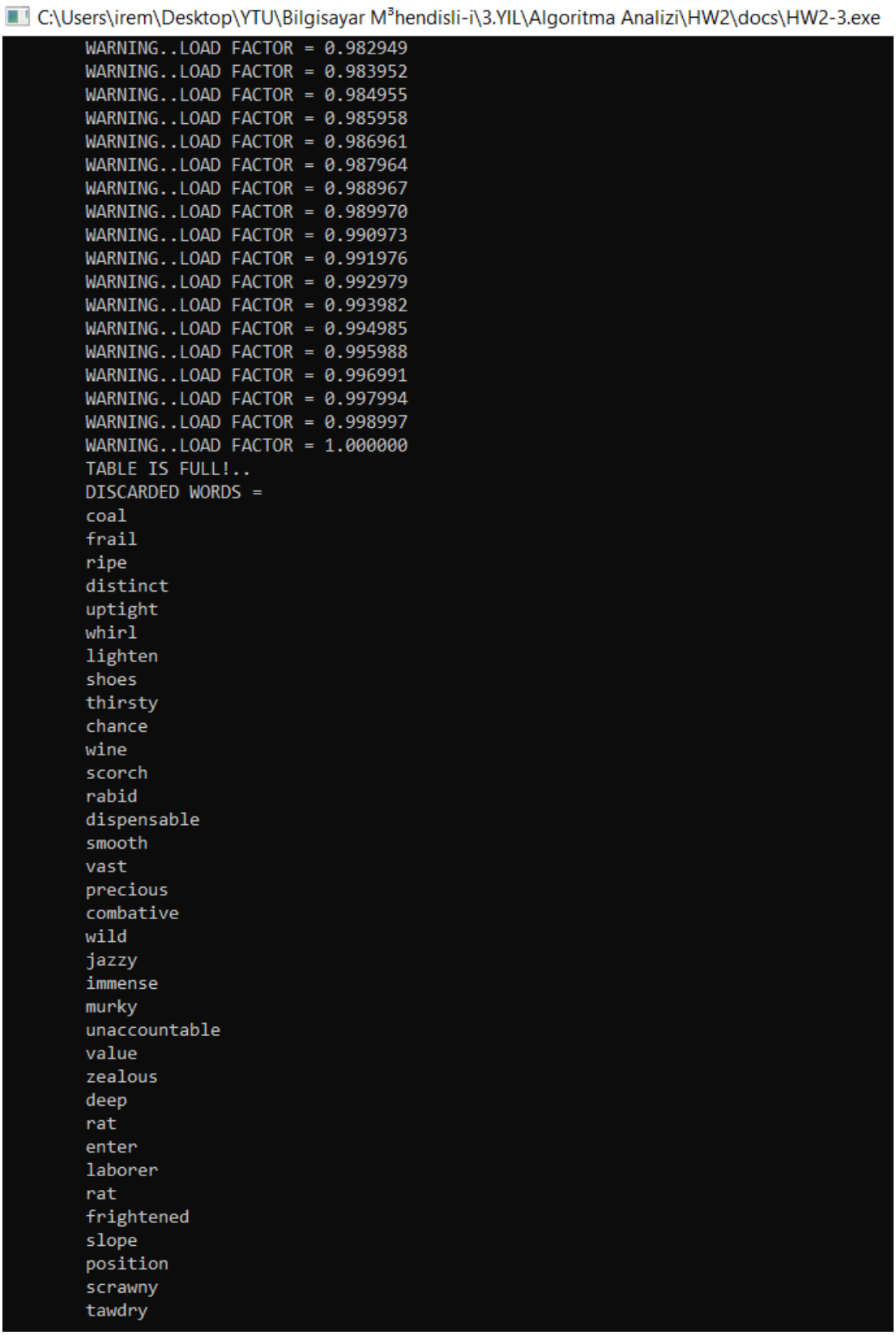
```
389 //Hash tablosunu dosyaya yazan fonksiyon
390 void updateFile(FILE* fp, HashTable* hashTable)
391 {
392     clock_t begin = clock();
393     char** tmp; //Geçici olarak döküman adlarını tutan string dizisi
394     short cursor;   //Döküman string'i içerisinde ilerlerken kullanacağımız indis değişkeni
395     float loadF = (float)(hashTable->N)/(hashTable->M); //load factor hesaplanır
396     fprintf(fp,"%lf\n",loadF); //İlk satırda load factor yer alır
397     fprintf(fp,"%d %d\n",hashTable->N,hashTable->M); //Dosyanın ikinci satırına tablodaki kelime sayısı ve tablo boyutu yazdırılır
398 
399     int i;
400     for(i = 0 ; i < TABLE_SIZE; i++)
401     {
402         //Hücre boş değilse dosyaya hücrenin indisi, kelimenin Horner's Metodu ile hesaplanmış anahtar değeri ve
403         //kelimeyi içeren dökümanlar sırasıyla satır satır dosyaya yazdırılır
404         if(hashTable->cells[i].hashedWord != -1)
405         {
406             tmp = hashTable->cells[i].docs;
407             fprintf(fp,"%d %d",i,hashTable->cells[i].hashedWord);
408             for(cursor = 0; cursor < hashTable->cells[i].document_cursor; cursor++)
409                 fprintf(fp," %s",tmp[cursor]);
410 
411             fprintf(fp,"\n");
412         }
413 
414     }
415     printf("\n\tWRITING COMPLETED!..");
416     clock_t end = clock();
417     double time_spent = (double)(end-begin) / CLOCKS_PER_SEC;  //fonksiyonun çalışma süresini yazdır
418     printf("\n\tWRITING TIME = %lf\n",time_spent);
419 }
```

EKRAN ÇIKTILARI

Örnek 1 : doc2.txt dosyasının hash tablosuna yerleştirilmesi

```
=== MAIN MENU ===
Options :
1. If you want to insert a new word please type 0
2. If you want to search for a word please type 1

Press ESC to quit

Case 1
Insert document name here : doc2

Document Name = doc2.txt
READING TABLE!..
READING IS DONE!..
LOAD FACTOR = 0.192578
LOAD FACTOR = 0.193581
LOAD FACTOR = 0.194584
LOAD FACTOR = 0.195587
LOAD FACTOR = 0.196590
LOAD FACTOR = 0.197593
LOAD FACTOR = 0.198596
LOAD FACTOR = 0.199599
LOAD FACTOR = 0.200602
LOAD FACTOR = 0.201605
LOAD FACTOR = 0.202608
LOAD FACTOR = 0.203611
LOAD FACTOR = 0.204614
LOAD FACTOR = 0.205617
LOAD FACTOR = 0.206620
LOAD FACTOR = 0.207623
LOAD FACTOR = 0.208626
LOAD FACTOR = 0.209629
LOAD FACTOR = 0.210632
LOAD FACTOR = 0.211635
LOAD FACTOR = 0.212638
LOAD FACTOR = 0.213641
LOAD FACTOR = 0.214644
LOAD FACTOR = 0.215647
LOAD FACTOR = 0.216650
LOAD FACTOR = 0.217653
LOAD FACTOR = 0.218656
LOAD FACTOR = 0.219659
LOAD FACTOR = 0.220662
LOAD FACTOR = 0.221665
LOAD FACTOR = 0.340020
LOAD FACTOR = 0.341023
LOAD FACTOR = 0.342026
LOAD FACTOR = 0.343029
LOAD FACTOR = 0.344032
LOAD FACTOR = 0.345035
LOAD FACTOR = 0.346038
LOAD FACTOR = 0.347041
LOAD FACTOR = 0.348044
LOAD FACTOR = 0.349047
LOAD FACTOR = 0.350050
LOAD FACTOR = 0.351053
LOAD FACTOR = 0.352056
LOAD FACTOR = 0.353059
LOAD FACTOR = 0.354062
LOAD FACTOR = 0.355065
LOAD FACTOR = 0.356068
LOAD FACTOR = 0.357071
LOAD FACTOR = 0.358074
LOAD FACTOR = 0.359077
LOAD FACTOR = 0.360080
INSERTION TIME = 1.156000

WRITING COMPLETED!..
WRITING TIME = 0.001000

=== MAIN MENU ===
Options :
1. If you want to insert a new word please type 0
2. If you want to search for a word please type 1

Press ESC to quit

--------------------------------
Process exited after 34.68 seconds with return value 0
Press any key to continue . . .
```

Örnek 2 : Load factor'ü 0.8'den 1'e çıkan hash table'a yapılan insertion işlemlerinin sonucu



```
        whisper
        field
        unsuitable
        subsequent
        nappy
        twist
        frantic
        five
        land
        flagrant
        piquant
        perform
        slope
        extra
        small
        dashing
        contain
        camera
        mere
        dry
        immense
        rude
        bedroom
        brother
        numberless
        onerous
        animated
        spooky
        bless
        knowledgeable
        pathetic
        connect
        snotty
        aunt
        lumpy
        unbecoming
        curved
        nose
        curly
        finger
        snakes
        afternoon
        tacit
        parallel
        glove
        === MAIN MENU ===
        Options :
        1. If you want to insert a new word please type 0
        2. If you want to search for a word please type 1

        Press ESC to quit

--------------------------------
Process exited after 91.47 seconds with return value 0
Press any key to continue . . .
```

Örnek 3 : Hash Table'a eklenen keliemeler arasından "thread" kelimesini içeren dökümanların bulunması

```
=== MAIN MENU ===
Options :
1. If you want to insert a new word please type 0
2. If you want to search for a word please type 1

Press ESC to quit
Case 2
Insert word here : thread

READING TABLE!..
READING IS DONE!..
WORD FOUND!..TOTAL STEPS = 6
IN DOCUMENTS : doc5

=== MAIN MENU ===
Options :
1. If you want to insert a new word please type 0
2. If you want to search for a word please type 1

Press ESC to quit
```