ALGORİTMA ANALİZİ

ÖDEV 3 - GRUP 1

İrem ATILGAN

17061036

20.12.2020

SORU 2

2.1 Fonksiyonlar

HashTable* insertTable(char*,HashTable*) : Hash tablosuna sözlük dosyasını ve sözlükteki kelimeleri yerleştiren fonksiyon. Parametre olarak hash tablosunu ve dosya içerisindeki metni alır, yerleştirilmiş tabloyu döner.

HashTable* createTable() : HashTable oluşturma fonksiyonu.

char *readDocument(char*,char*) : Sözlük dosyasını okuma fonksiyonudur. Dosyada yer alan metni döner.

int search(char*,HashTable*) : Verilen bir hash tablosunda kelime arama fonksiyonu. Parametre olarak hash tablosu ve aranan kelime verilir; Eğer kelime bulunamazsa -1, bulunursa kelimenin tablodaki adresi dönülür.

long int hornersMethod(char*,int) : Verili kelimeden anahtar değer hesaplayan Horner's Yöntemi'dir. Parametre olarak kelimeyi ve kelime uzunluğunu alır, hesaplanan anahtar değeri döner.

int create2DMatrix(int,int) :** 2 Boyutlu matris oluşturan fonksiyondur.

char* recommendWords(char*,HashTable*) : Levenshtein Edit Distance algoritmasi ile kullanıcılara hatalı kelime için önerilebilir kelimeler sunan fonksiyondur. Parametre olarak yanlış girilen kelimeyi ve sözlük kelimelerinin olduğu hash tablosu alınır. Kullanıcının tercih ettiği önerilen kelime dönülür.

HashTable* insertCrookedTable(char*,char*,HashTable*) : Hatalı kelimelerin doğrularının yer aldığı hash tablosuna kelime yerleştirme fonksiyonudur. Parametre olarak hatalı kelime, kullanıcının tercih ettiği önerilen kelime ve hash tablosu alınır.

int findDif(char,char) : Kelime içerisindeki harflerin aynı olup olmadığını kontrol eden fonksiyondur. Levenshtein E.D. Algoritması için oluşturulmuştur. Parametre olarak iki harf alır; harfler aynıysa sıfır, değilse bir (1) döner.

int findMin(int,int,int) : L.E.D. algoritmasında hesaplanan 3 seçenek içerisinden minimum değeri bulan fonksiyondur.

void printHT(HashTable*) : Hash tablosunu yazdırma fonksiyonudur.

char* convertUpperCase(char*) : Kelimenin harflerini upper case'e çeviren fonksiyondur. Parametre olarak kelime alınır, çevrilmiş hali dönülür.

int LED(char*,char*) : Levenshtein Edit Distance Algoritması'nın yer aldığı fonksiyondur. Fonksiyonda matris oluşturularak dinamik programlama ile parametre olarak alınan kelimeler arasındaki mesafe bulunur. Mesafe değeri dönülür.

Double Hashing yöntemi için kullanılan iki hash fonksiyonu da aşağıda verilmiştir. Horner's Yöntemi ile hesaplanan anahtar değer bu fonksiyonlara parametre olarak verilir ve kelimenin hash tablosunda yerleşeceği adres hesaplanır.

int hash1(int);

int hash2(int);

2.2 Program Kodu

2.2.1 Structlar

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

//HashTable struct'ında yer alan hücreler ayrı bir structta tutulur
typedef struct CELL{
    char word[20];       //Tutulan kelime
    long int hashedWord; //Horner's metodu ile kelimeden elde edilen anahtar değer
}CELL;

//Hash tablosu struct'ı
typedef struct HashTable{
    CELL* cells; //Tabloda kelimeleri tutan hücreler
}HashTable;
```

2.2.2 Fonksiyonların Tanımlanması

```c
HashTable* insertTable(char*,HashTable*);       //Hash tablosuna döküman ve döküman kelimelerini yerleştiren fonksiyon
HashTable* createTable();                       //HashTable struct'ını başlatan (initializing) fonksiyon
char *readDocument(char*,char*);                //Sozluk dosyasını okuma fonksiyonu
int search(char*,HashTable*);                   //HashTable struct'ında kelime arama fonksiyonu
long int hornersMethod(char*,int);              //Verili kelimeden anahtar değer hesaplayan Horner's Yöntemi
int hash1(int);
int hash2(int);
int** create2DMatrix(int,int);                  //2 Boyutlu matris olusturan fonksiyon
char* recommendWords(char*,HashTable*);         //Levenshtein Edit Distance algoritmasi ile kullanicilara hatali kelime icin onerilebilir kelimeler sunan fonksiyon
HashTable* insertCrookedTable(char*,char*,HashTable*);  //Hatalı kelimelerin dogrularinin yer aldigi hash tablosuna kelime yerlestirme fonksiyonu
int findDif(char,char);                         //Parametre olarak verilen harflerin ayni olup olmadigini kontrol eden fonksiyon
int findMin(int,int,int);                       //LED algoritmasinda 3 secenek icerisinden minimum degeri bulan fonksiyon
void printHT(HashTable*);                       //Hash tablosunu yazdirma fonksiyonu
char* convertUpperCase(char*);                  //Kelime harflerini upper case'e ceviren fonksiyon
int LED(char*,char*);                           //Levenshtein Edit Distance Algoritmasi
```

2.2.3 Create2DMatrix & findDif & findMin Fonksiyonları

```
36  //2 Boyutlu matris olusturan fonksiyon
37  int** create2DMatrix(int dim1,int dim2)
38  {
39      int i;
40      int **matrix;
41      matrix = calloc(dim1,sizeof(int*));
42      for(i = 0; i < dim1; i++)
43      {
44          matrix[i] = calloc(dim2,sizeof(int));
45      }
46
47      return matrix;
48  }
49
50  //Parametre olarak verilen harflerin ayni olup olmadigini kontrol eden fonksiyon
51  int findDif(char a, char b)
52  {
53      if(a > 96)  a = a - ('a'-'A');
54      if(b > 96)  b = b - ('b'-'B');
55      if(a-b) return 1;
56      return 0;
57  }
58
59  //LED algoritmasinda 3 secenek icerisinden minimum degeri bulan fonksiyon
60  int findMin(int opt1, int opt2, int opt3)
61  {
62      if(opt1 < opt2 && opt1 < opt3)  return opt1;
63      else
64      {
65          if(opt2 < opt1 && opt2 < opt3)  return opt2;
66          else    return opt3;
67      }
68  }
```

2.2.4 Hash Fonksiyonları ve hornersMethod Fonksiyonu

```
71  int hash1(int key)
72  {
73      return key%(TABLE_SIZE);
74  }
75
76  int hash2(int key)
77  {
78      return (1+(key%(TABLE_SIZE-1)));
79  }
80
81  //Verili kelimeden anahtar değer hesaplayan Horner's Yöntemi
82  long int hornersMethod(char* word, int length)
83  {
84      long int key = 0;
85      int i;
86      int r = 31; //asal sayı
87
88      i = length;
89      while(i >= 0)
90      {
91          key += word[i]*pow(r,i);
92          key = key & 0x7FFFFFFF; //Overflow olmasını engellemek için
93          i--;
94      }
95
96      return key;
97  }
```

2.2.5 readDocument Fonksiyonu

```
71  int hash1(int key)
72  {
73      return key%(TABLE_SIZE);
74  }
75
76  int hash2(int key)
77  {
78      return (1+(key%(TABLE_SIZE-1)));
79  }
80
81  //Verili kelimeden anahtar değer hesaplayan Horner's Yöntemi
82  long int hornersMethod(char* word, int length)
83  {
84      long int key = 0;
85      int i;
86      int r = 31; //asal sayı
87
88      i = length;
89      while(i >= 0)
90      {
91          key += word[i]*pow(r,i);
92          key = key & 0x7FFFFFFF; //Overflow olmasını engellemek icin
93          i--;
94      }
95
96      return key;
97  }
```

2.2.6 convertUpperCase Fonksiyonu

```
176 //Kelime harflerini upper case'e ceviren fonksiyon
177 char* convertUpperCase(char* word)
178 {
179     int i;
180     char tmp_word[20];
181
182     char* temp = calloc(20,sizeof(char));
183     for(i = 0; word[i] != '\0'; i++)
184     {
185
186         if(word[i] > 96)    //lower case harf ise
187             temp[i] = word[i] - ('a'-'A');
188         else
189             temp[i] = word[i];
190
191     }
192     temp[i] = '\0';
193
194     return temp;
195 }
```

2.2.7 search Fonksiyonu

```
197  //HashTable struct'ında kelime arama fonksiyonu
198  int search(char* word, HashTable* ht)
199  {
200      long int key;       //Horner's Metodu ile hesaplayacağımız anahtar değerini tutan değişken
201      int hash;           //Hashing sonucu hesaplanacak adres degeri
202      int k = 0;          //Collision olma durumunda yeni adresi bulurken düzenli olarak arttıracağımız değişken
203      char* converted;    //Upper case'e donusecek hatali kelime
204
205      //Aranan kelimenin harfleri upper case yapilir
206      converted = convertUpperCase(word);
207      //Horner's Yöntemi ile kelimenin anahtarı oluşturulur
208      key = hornersMethod(converted,strlen(word));
209      //Double hashing ile yeni adres hesaplanır
210      hash = (hash1(key) + k*hash2(key))%(TABLE_SIZE);
211
212      //Tum tablo dolasilana ya da aranan kelime bulunana kadar tabloda arama yapilir
213      while(k < TABLE_SIZE && ht->cells[hash].hashedWord != key)
214      {
215          key = hornersMethod(converted,strlen(word));
216          hash = (hash1(key) + k*hash2(key))%(TABLE_SIZE);
217          k++;
218      }
219
220      //Kelime bulunursa
221      if(k < TABLE_SIZE && ht->cells[hash].hashedWord == key)
222      {
223          printf("\tWord has been found!..%s\n",ht->cells[hash].word);
224          return hash;
225      }
226
227      return 0;   //Kelime bulunamazsa
228  }
```

2.2.8 insertTable Fonksiyonu

```
405  //HashTable struct'ina metin yerlestiren fonksiyon
406  HashTable* insertTable(char* text, HashTable* ht)
407  {
408
409      int i = 0;
410      int m = 0;  //Okunan kelimenin index'ini tutan degisken
411
412      int k;              //Collision olma durumunda yeni adresi bulurken düzenli olarak arttıracağımız değişken
413      long int key;       //Horner's Metodu ile hesaplayacağımız anahtar değerini tutan değişken
414      int hash;           //Hashing sonucu hesaplanacak adres degeri
415      char word[20];      //Metinde okunan her kelimeyi gecici olarak tutan char dizisi
416      char* converted;    //Upper case'e donusecek hatali kelime
417      for(i = 0; text[i] != '\0'; i++)
418      {
419          if(text[i] == ' ')
420          {
421              word[m] = '\0';
422              converted = convertUpperCase(word);
423
424              //Horner's Yöntemi ile kelimenin anahtarı oluşturulur
425              key = hornersMethod(converted,strlen(converted));
426              //Double hashing ile yeni adres hesaplanır
427              hash = (hash1(key) + k*hash2(key))%(TABLE_SIZE);
428
429              k = 0;
430              //Tüm tablo dolaşılmadığı, bulunan adres boş olmadığı ve bulduğumuz adresteki kelime ile aradığımız kelime eşleşmediği sürece
431              //k değişkenini arttırmaya ve yeni adres değerini hesaplamaya devam et
432              while(k < TABLE_SIZE && ht->cells[hash].hashedWord != NULL && ht->cells[hash].hashedWord != hash)
433              {
434                  key = hornersMethod(converted,strlen(converted));
435                  hash = (hash1(key) + k*hash2(key))%(TABLE_SIZE);
436                  k++;
437              }
438              //Boş yer ya da aynı kelime bulunmuşsa
439              if(k < TABLE_SIZE)
440              {
441                  //Boş hücreye yerleştirilecek kelimenin anahtar değeri, ve kelime yerleştirilir
442                  ht->cells[hash].hashedWord = key;
443                  strncpy(ht->cells[hash].word,converted,sizeof(ht->cells[hash].word));
444              }
```

```
            m = 0;
        }
        else
            word[m++] = text[i];
    }

    return ht;
}
```

2.2.9 insertCrookedTable Fonksiyonu

```
//Hatalı kelimelerin dogrularinin yer aldigi hash tablosuna kelime yerlestirme fonksiyonu
HashTable* insertCrookedTable(char* right_word, char* crooked_word, HashTable* cw)
{
    char* converted;        //Upper case'e donusecek hatali kelime
    char* right_converted;  //Upper case'e donusecek dogru kelime
    int k = 0;              //Collision olma durumunda yeni adresi bulurken düzenli olarak arttıracağımız değişken
    long int key;           //Horner's Metodu ile hesaplayacağımız anahtar değerini tutan değişken
    int hash;               //Hashing sonucu hesaplanacak adres degeri

    //Dogru ve hatali kelime upper case hale getirilir
    right_converted = convertUpperCase(right_word);
    converted = convertUpperCase(crooked_word);

    //Kelimeden anahtar çıkarılır
    key = hornersMethod(converted,strlen(converted));
    //Double hashing ile yeni adres hesaplanır
    hash = (hash1(key) + k*hash2(key))%(TABLE_SIZE);

    //Hatali kelimeden hesaplanan adres, collision olmayana dek yeniden hesaplanır
    while(k < TABLE_SIZE && cw->cells[hash].hashedWord != NULL && cw->cells[hash].hashedWord != hash)
    {
        key = hornersMethod(converted,strlen(converted));
        hash = (hash1(key) + k*hash2(key))%(TABLE_SIZE);
        k++;
    }
    //Eger tabloda bos yer bulunmussa
    if(k < TABLE_SIZE)
    {
        cw->cells[hash].hashedWord = key; //Hatali kelimenin adresine hatali kelime icin uretilen anahtar deger yerlestirilir
        strncpy(cw->cells[hash].word,right_converted,sizeof(cw->cells[hash].word)); //Kelimenin dogrusu adrese eklenir
    }
    return cw;
}
```

2.2.10 recommendWords Fonksiyonu

```
//Levenshtein Edit Distance algoritmasi ile kullanicilara hatali kelime icin onerilebilir kelimeler sunan fonksiyon
char* recommendWords(char* word, HashTable* dict)
{
    int i;
    int cursor = 0;         //Onerilebilir kelime listesi index'i
    int dist;               //Kelimelerin mesafelerini (edit distance) tutan degisken
    int lim = 2;            //Distance limiti 2 olarak belirlenmistir
    char new_word[20];      //Kullanicinin onerilecek kelimeler arasindan sectigi kelimeyi tutacak char dizisi

    //Onerilen kelimeler listesi initialize edilir
    char** word_list = (char**)calloc(100,sizeof(char*));
    for(i = 0; i < 100; i++)
        word_list[i] = (char*)calloc(20,sizeof(char));

    //Hash tablosundaki her kelime gezilerek hatali kelimeyle mesafeleri hesaplanir
    for(i = 0; i < TABLE_SIZE; i++)
    {
        //printf("i = %d\n",i);
        if(strlen(dict->cells[i].word) != 0)    //Hash tablosunun bos hucresi degilse
        {
            dist = LED(word,dict->cells[i].word);   //Levenshtein Edit Distance hesaplanir
            //Eger limitten az mesafe bulunduysa ve minimum mesafeye esit ya da kucukse oneri listesine eklenir
            if(dist <= lim)
            {
                strcpy(word_list[cursor],dict->cells[i].word);
                cursor++;
            }
        }

    }
```

```
319     printf("\n");
320     if(cursor == 0) //Listeye uygun hicbir kelime eklenemezse NULL donulur
321     {
322         printf("\tThere are no words can be recommended!..Please try again\n");
323         return NULL;
324     }
325     //Onerilebilir kelimeler kullaniciya sunulur
326     printf("\tDid you mean : ");
327     for(i = 0; i < cursor-1; i++){
328         printf(" \"%s\" or",word_list[i]);
329     }
330     printf(" \"%s\"\n\t>>",word_list[i]);
331     scanf("%s",new_word);   //Kullanicidan tercih ettigi, onerilen kelime alinir
332 
333     return new_word;
334 }
```

2.2.11 LED (Levenshtein Edit Distance) Fonksiyonu

```
157 //Levenshtein Edit Distance Algoritmasi
158 int LED(char* ref, char* old)
159 {
160     int i;
161     int j;
162     //Aralarindan minimumu bulunacak olan 3 uzakligin tutuldugu degisken
163     int opt1,opt2,opt3;
164     int M = strlen(ref);
165     int N = strlen(old);
166     int** mat = create2DMatrix(M+1,N+1); //Dinamik programlama icin matris olusturulur
167     i = 0;
168     while(i <= M)
169     {
170         for(j = 0; j <= N; j++)
171         {
172             if(i == 0 && j == 0)
173                 mat[i][j] = 0;
174             else if(j == 0)
175                 mat[i][j] = mat[i-1][j] + 1;
176             else if(i == 0)
177                 mat[i][j] = mat[i][j-1] + 1;
178             else
179             {
180                 opt1 = mat[i-1][j] + 1;
181                 opt2 = mat[i][j-1] + 1;
182                 opt3 = mat[i-1][j-1] + findDif(ref[i-1],old[j-1]);  //harflerin ayni olup olmadigi hesaplanarak opt3 e eklenir
183                 mat[i][j] = findMin(opt1,opt2,opt3);                //Seceneklerden minimum distance'i veren deger secilir
184             }
185         }
186         i++;
187     }
188     return mat[M][N];   //Sonuc distance'i donulur
189 }
```

2.2.12 main Fonksiyonu

```
int main()
{
    char text[10000];   //Verilen smalldictionary.txt dosyası icerigini tutan char dizisi
    char sentence[50];      //Aranacak kelimenin tutuldugu degisken
    char* result_text;  //Okunan dosya iceriginin yer aldigi char dizisinin adresini tutan pointer degisken
    char* right_word;   //recommendWords fonksiyonu ile kullanıcının sectigi onerilen kelimeyi tutan degisken
    char *splitted_word; //strtok işlemleri için kullanılan geçici char* değişkeni
    char newstr[20];    //strtok ile elde edilen kelimeleri tutan char dizisi
    int flag = 1;       //Programın devamlılıgını saglayan bayrak degeri tutan degisken
    result_text = readDocument("smalldictionary.txt",text); //Dosyadaki icerik okunut
    HashTable* dictionary = createTable();  //Hash Tablosu olusturulur
    dictionary = insertTable(text,dictionary);  //Okunan kelimeler hash tablosuna eklenir
    HashTable* crookedWords = createTable();    //Hatali kelimelerin yer aldigi hash tablosu olusturulur
    do
    {
        printf("\n\t=== MAIN MENU ===");
        printf("\n\tOptions : \n");
        printf("\tIf you want to search for a word please type : ");
        gets(sentence);
        splitted_word = strtok(sentence," ");
        if(splitted_word != NULL)   //Kullanicinin girdigi kelime okunur, input alınıp alinmadigi kontrol edilir
        {
            while(splitted_word != NULL)
            {
                strcpy(newstr,splitted_word);
                if((!search(newstr,dictionary)) && (!search(newstr,crookedWords)))  //Sozluk tablosunda ve hatali kelime tablosunda kelimenin olup olmadigina bakilir
                {
                    //Kelime hicbir yerde yer almıyorsa
                    //Levenshtein algoritmasi ile hatali kelimeye en yakın kelimeler kullaniciya onerilir ve kullanicinin tercih ettigi kelime geri donulur
                    right_word = recommendWords(newstr,dictionary);
                    if(right_word != NULL) //Onerilebilir kelime varsa
                    {
                        //Dogru kelime hatali kelime tablosuna yerlestirilir
                        crookedWords = insertCrookedTable(right_word,newstr,crookedWords);
                    }
                }
                splitted_word = strtok(NULL," ");
            }
            getchar(); //Kullanıcının önerilen kelimeyi yazmasıyla gelen '\n' karakteri getchar ile alınır
        }
        else{
            printf("\n\tInvalid Input!.."); //Eger kullanicidan input alinmazsa program durdurulur
            flag = 0;
        }

    }
    while(flag);

    return 0;
}
```

2.3.5 Ekran Çıktısı

```
=== MAIN MENU ===
Options :
If you want to search for a word please type : happy new year

happy
There are no words can be recommended!..Please try again

new
Did you mean :  "ONE" or "NEXT" or "NO" or "GET" or "BE" or "WE" or "HOW" or "LOW" or "NEED"
>>next

year
Did you mean :  "YOUR" or "HEAD" or "READ" or "DEAL"
>>deal

=== MAIN MENU ===
Options :
If you want to search for a word please type : happy new year

happy
There are no words can be recommended!..Please try again
NEXT
Word has been found!..
DEAL
Word has been found!..


=== MAIN MENU ===
Options :
If you want to search for a word please type : main

main
Did you mean :  "GAIN" or "CAN" or "AN" or "IN" or "CHAIN" or "MANY" or "FAIR" or "PAIR" or "HAIR" or "MAY"
>>gain

=== MAIN MENU ===
Options :
If you want to search for a word please type :

Invalid Input!..
```