

YILDIZ TEKNİK ÜNİVERSİTESİ

BİLGİSAYAR MÜHENDİSLİĞİ BÖLÜMÜ

2012-2013 GÜZ YARIYILI

ALGORİTMA ANALİZİ

3.ödev raporu

Ödev Konusu: Hashing

Öğrenci bilgileri

Ad: ONUR

Soyad: ÖNAL

Numara: 10011603

Gerçeklenen bu uygulamada harf yazım kontrolü (spell checker) işlemi yapan bir sistem tasarlanmıştır. Sistem, dışarıdan verilen bir kelimenin sözlükte olup olmadığını kontrol etmekte, kelime sözlükte bulunursa ‘kelime doğru yazılmıştır’ mesajı vermektedir. Kelime sözlükte bulunamazsa sözlükte bulunan kelimelerden doğru kelime önerileri yapılabiliyorsa kelime önerileri yapılmaktadır.

1. **Yöntem**

Uygulama hash tablosu oluşturma, verilen kelimeyi hash tablosunda arama ve kelime sözlükte yoksa kelime önerme olmak üzere 3 işlevden oluşmaktadır. İlk olarak kelimelerin saklandığı smallDictionary.txt isimli dosyadan kelimeler okunmakta ve hash tablosu oluşturulmaktadır. Hash tablosu oluşturulurken openaddress yöntemi, çakışma problemini çözmek için double hashing yöntemleri kullanılmaktadır. Hash tablosunun boyutu ise load factor’ün 0.7 olması için 487 olarak seçilmiştir. Sözlükte toplam 339 kelime yer aldığından 339/0.7 işlemi yapılmaktadır. Bu değerden büyük en küçük asal sayı da 487 olduğu için hash tablosunun boyutu 487 seçilmiştir. Bu yöntemlere göre hash tablosu oluşturulduktan sonra hashTable.txt isimli dosyaya kaydedilmektedir. Uygulama eğer bu isimde bir dosya yoksa hash tablosu oluşturacak şekilde tasarlandığı için bu işlem sadece 1 defa yapılmaktadır.

Kullanıcıdan alınan kelimenin sözlükte aranması aşamasında ise, ilk olarak hash fonksiyonu kullanılarak aranan kelimeden elde edilen h(key) adresi elde edilmektedir. Aranan kelimenin bu adreste olup olmadığına bakılmaktadır. Aranan eleman bu adreste değilse, hash tablosu oluşturulurken kullanılan double hashing yöntemine göre elemanın tabloda olup olmadığına bakılmaktadır. Kelime tabloda varsa tabloda hangi indislere bakıldıktan sonra o değere erişildiği bilgisi yani probe(erişim) sayısı kullanıcıya gösterilmektedir.

Kullanıcıdan alınan kelime sözlükte yok ise, yanlış kelime olduğuna karar verilmektedir. Yanlış kelime yerine kelime önerme işlemi için her seferinde verilen kelimedeki tek harfi değiştiren basit bir yöntem kullanılmaktadır. Bu yöntemde kelimedeki her harf sırası ile ‘a’ dan ‘z’ ye harflerle değiştirilerek elde edilen yeni kelimenin hash tablosunda olup olmadığına bakılmaktadır. Eğer elde edilen yeni kelime hash tablosunda varsa yanlış kelime yerine önerilen kelimelerden biri olarak ekrana yazdırılmaktadır.

Gerçeklenen bu uygulamaya ait pseudo code şöyledir;

**Pseudo code**

1. Diskte hash Tablosu var mı diye kontrol et.
2. Eğer yoksa openaddress yöntemine göre hash tablosunu oluştur, çakışma problemini double hashing yöntemlerine göre çöz. Tablo oluşturulduktan sonra diske kaydet. Eğer varsa 3. adımdan devam et
3. Diskten dosyayı oku, hash tablosunu oluştur.
4. Kullanıcıdan kelimeyi al.
5. Kelimenin h(key) adresini belirle.
6. Aranan kelime h(key) adresinde ise kullanıcıya erişim yolunu göster. Uygulamayı sonlandır.
7. Aranan kelime h(key) adresinde değilse tablo oluşturulurken kullanılan double hashing yöntemine göre bir sonraki olabilecek adresi belirle. Bu adres bulma işlemine aranan kelime bulunana kadar veya hash tablosunda boş bir adrese bakılana kadar devam et.
8. Eğer kelime bulunmuşsa kullanıcıya erişim yollarını göster. Uygulamayı sonlandır.
9. Eğer hash tablosunda boş bir adrese gelinmişse, bu kelime yerine her seferinde yalnızca bir harf değiştirerek elde edebileceğin yeni kelimeler üret.
10. Üretilen kelime için 5. adımdan itibaren aynı işlemleri 8. Adıma kadar tekrar et. Eğer üretilen kelime hash tablosunda bulunmuşsa kullanıcıya bu kelimeyi öner, bulunmamışsa bir sonraki kelimeyi üret.
11. Üretilebilecek tüm kelimeler için 10. adımı tekrar et. Uygulamayı sonlandır.
12. **Uygulama**

Bu bölümde uygulamaya sözlükte olan ve olmayan kelimeler verilerek sözlükte olan kelimelerin ortalama aranma süresinin ikili arama yönteminden daha hızlı olup olmadığı değerlendirilmiştir.

Sözlükte yer alan kelimelerden **algorithm**, **have**, **longest** ve **track** kelimeleri için değerlendirme yapılmıştır.

Değerlendirme hashing için hashTable.txt dosyasında yer alan hash tablosuna göre yapılmıştır. İkili arama yöntemi için smallDictionary.txt dosyasında kelimeler sıralı olduğundan smallDictionary.txt dosyasında yer alan sıraya göre yapılmıştır.

**algorithm** kelimesi için;

Hashing ile bu kelimeye erişilmek istenirse hash tablosunun 31. indise bakılır ve kelimeye erişilmiş olunur. Bu nedenle erişim sayısı 1’dir.

İkili arama yöntemiyle bu kelimeye erişilmek istenirse kelime 8. sırada yer aldığı için ve toplamda 339 kayıt yer aldığı için, ilk olarak (0+338/2) 169. kelimeye bakılır. Daha sonra (0+168/2) 84. kelimeye bakılır. Ardından sırasıyla 41. kelimeye, 20. kelimeye, 9. kelimeye, 4. kelimeye, 6. kelimeye,7. kelimeye, 8. kelimeye bakılır ve kelimeye erişilmiş olunur. Bu nedenle erişim sayısı 9 ‘dur.

**have** kelimesi için;

Hashing ile bu kelimeye erişilmek istenirse hash tablosunun 430. indise bakılır ve kelimeye erişilmiş olunur. Bu nedenle erişim sayısı 1’dir.

İkili arama yöntemiyle bu kelimeye erişilmek istenirse kelime 132. sırada yer aldığı için ve toplamda 339 kayıt yer aldığı için sırasıyla 169. kelimeye, 84. kelimeye, 126. kelimeye, 147. kelimeye, 136. kelimeye, 131. kelimeye, 133. kelimeye, 132. kelimeye bakılır ve kelimeye erişilmiş olunur. Bu nedenle erişim sayısı 8’dir.

**longest** kelimesi için;

Hashing ile bu kelimeye erişilmek istenirse hash tablosunun sırası ile 410. indise, 348. indise bakılır ve kelimeye erişilmiş olunur. Bu nedenle erişim sayısı 2’dir.

İkili arama yöntemiyle bu kelimeye erişilmek istenirse kelime 172. sırada yer aldığı için ve toplamda 339 kayıt yer aldığı için sırasıyla 169. kelimeye, 254. kelimeye, 211. kelimeye, 190. kelimeye, 179. kelimeye, 174. kelimeye, 171. kelimeye, 172. kelimeye bakılır ve kelimeye erişilmiş olunur. Bu nedenle erişim sayısı 8’dir.

**track** kelimesi için;

Hashing ile bu kelimeye erişilmek istenirse hash tablosunun 177. indise bakılır ve kelimeye erişilmiş olunur. Bu nedenle erişim sayısı 1’dir.

İkili arama yöntemiyle bu kelimeye erişilmek istenirse kelime 292. sırada yer aldığı için ve toplamda 339 kayıt yer aldığı için sırasıyla 169. kelimeye, 254. kelimeye, 296. kelimeye, 275. kelimeye, 285. kelimeye, 290. kelimeye, 293. kelimeye, 292. kelimeye bakılır ve kelimeye erişilmiş olunur. Bu nedenle erişim sayısı 8’dir.

Bu yapılan küçük incelemeden de anlaşılabileceği gibi hashing ile bir kayda erişmek için geçen süre ikili arama yöntemiyle bir kayda erişmek için geçen süreden çok çok azdır. Teoride de zaten hashing’de erişim sayısı 1’dir. İkili arama yönteminde ise ortalama erişim sayısı ’dir.

Sözlükte yer almayan kelimelerden ise **anc**, **ik**, **sare** ve **kave** kelimeleri için değerlendirme yapılmıştır.

**anc** kelimesi için;

Hashing ile bu kelimeye erişilmek istenirse hash tablosunun 442. indise bakılmıştır. Bu indisde başka bir kelime yer aldığından 348. indise bakılmıştır. 348. indis boş olduğu için bu kelimenin sözlükte olmadığı anlaşılmıştır. Bu kelime için sözlükte yer alan **and** ve **any** kelimeleri önerilmiştir.

**ik** kelimesi için;

Hashing ile bu kelimeye erişilmek istenirse hash tablosunun 260. indise bakılmıştır. Bu indisde başka bir kelime yer aldığından 34. indise bakılmıştır. 34. indis boş olduğu için bu kelimenin sözlükte olmadığı anlaşılmıştır. Bu kelime için sözlükte yer alan **if, in, is** ve **it** kelimeleri önerilmiştir.

**sare**  kelimesi için;

Hashing ile bu kelimeye erişilmek istenirse hash tablosunun sırası ile 222. indise, 447. indise, 185. indise, 410. indise, 148. indise, 373. indise, 111. indise, 336. indise, 74. indise ve 299. indise bakılmıştır. 299. indis boş olduğu için bu kelimenin sözlükte olmadığı anlaşılmıştır. Bu kelime için sözlükte yer alan **same** ve **save** kelimeleri önerilmiştir.

**Kave** kelimesi için;

Hashing ile bu kelimeye erişilmek istenirse hash tablosunun sırası ile 430. indise, 376. indise, 322. indise, 268. indise, 214. indise, 160. indise, 106. indise, 52. indise, 485. indise, 431. indise, 377. indise, 323. indise, 269. indise, 215. indise ve 161. indise bakılmıştır. 161. indis boş olduğu için bu kelimenin sözlükte olmadığı anlaşılmıştır. Bu kelime için sözlükte yer alan **have** ve **save** kelimeleri önerilmiştir.

1. **Kaynak Kod**

#include <stdio.h>

#include <stdlib.h>

//Eger hashTable.txt dosyasi olusturulmamissa bu dosyayi olusturan fonksiyon

//hashTable.txt dosyasi silinmedigi surece sadece 1 kere islem yapmaktadir

void createHashTable(char \*\*hashTable,char \*\*dictionary)

{

FILE \*fp;//file pointer

char x;//EOF karakterine gelindigi bilgisini tutar

int i,j,k;//index degiskenleri

int key;//hash fonksiyonunun uretecegi key degerini tutar

int h1,h2;//hashing fonksiyonlarinin degerlerini tutarlar

int MM,M;//hash tablosunun uzunlugunu tutar

M=487;//load factor=0.7 kabul edildiginde tablo uzunlugu en kucuk asal sayi 487 olur

MM=M-1;

fp = fopen("hashTable.txt", "r");

if (fp == NULL)

{

//hashTable.txt dosyasi eger olusturulmamissa bu islemleri yapar.Yani sadece 1 defa //calismasi icin yapilmistir

fp = fopen("smallDictionary.txt", "r");

if (fp == NULL) {

printf("I couldn't open smallDictionary.txt\n");

exit(0);

}

i=0;

do{

x=fscanf(fp,"%s",dictionary[i]);//sozlukte 339 kayit var

i++;

}while(x!=EOF);

for(i=0;i<339;i++)//sozlukte 339 kayit oldugu icin i<339

{

key=0;

for(j=0;j<strlen(dictionary[i]);j++)

{

key+=(dictionary[i][j]-'a')\*26\*j;

}

h1=key%M;

h2=1+(key%MM);

j=0;

while( hashTable[(h1+j\*h2)%M][0]!=NULL)//tablonun bakilan hucresi dolu ise bos //hucre arar

{

j++;

}

///////////////////////////////////////////////sozlukteki kelimeyi hash tablosuna kopyalar

k=0;

while(dictionary[i][k]!='\0')

{

hashTable[(h1+j\*h2)%M][k]=dictionary[i][k];

k++;

}

hashTable[(h1+j\*h2)%M][k]=dictionary[i][k];

////////////////////////////////////////////////

}

fclose(fp);

fp = fopen("hashTable.txt", "w");

if (fp == NULL) {

printf("I couldn't write hashTable.txt\n");

exit(0);

}

for(i=0;i<487;i++)

{

if(hashTable[i][0]==NULL)

fprintf(fp,"# ");//hash table da bos hucreler icin # sembolu yazilmistir

else

fprintf(fp,"%s ",hashTable[i]);

}

fclose(fp);

printf("hashTable.txt didn't create.I create it.\n");

}

}

//hashTable isimli string dizisine hashTable.txt isimli dosyada yer alan kelimeleri atar

void readHashTable(char \*\*hashTable)

{

FILE \*fp;//file pointer

char x;//EOF karakterine gelindigi bilgisini tutar

int i;//index degiskeni

fp = fopen("hashTable.txt", "r");

if (fp == NULL) {

printf("I couldn't open hashTable.txt\n");

exit(0);

}

i=0;

do{

x=fscanf(fp,"%s",hashTable[i]);

i++;

}while(x!=EOF);

fclose(fp);

}

//parametre olarak aktarilan kelimenin yine parametre olarak aktarilan hash tablosunda olup //olmadigini bulur

//Sonuc olarak hash tablosunda aranan kelime varsa 1, yoksa -1 dondurur

int find(char kelime[],char \*\*hashTable)

{

int i,j;//index degiskenleri

int key;//hash fonksiyonunun uretecegi key degerini tutar

int h1,h2;//hashing fonksiyonlarinin degerlerini tutarlar

int MM,M;//hash tablosunun uzunlugunu tutar

int found=0;//sonucun donduruldugu bayrak degiskeni

M=487;//load factor=0.7 kabul edildiginde tablo uzunlugu en kucuk asal sayi 487 olur

MM=M-1;

key=0;

j=0;

for(i=0;i<strlen(kelime);i++)

{

key+=(kelime[i]-'a')\*26\*i;

}

while(found==0)

{

h1=key%M;

h2=1+(key%MM);

printf("h1=%d h2= %d key=%d \n",h1,h2,(h1+j\*h2)%M);

if(strcmp(hashTable[(h1+j\*h2)%M],kelime)==0)

found=1;

else if(strcmp(hashTable[(h1+j\*h2)%M],"#")==0)//hash tablosunda # karakteri ile //karsilasilmissa

found=-1; //daha ilerisine bakmaya gerek yok, kelime yoktur

j++;

}

return found;

}

//Parametre olarak aktarilan kelime parametre olarak aktarilan hash tablosunda varsa kelimeye //erisim yollarini

//yazar.Eger yoksa hash tablosunda onerilebilecek kelime varsa onlari onerir.

void search(char kelime[],char \*\*hashTable)

{

int found=0;

found=find(kelime,hashTable);

if(found==-1)

{

printf("Aranilan Kelime Sozlukte Yok\n Onerilen kelimeler:\n");

suggestion(hashTable,kelime);

}

else

{

printf("Aranilan Kelime Sozlukte Var\n");

}

}

void suggestion(char \*\*hashTable,char kelime[])

{

char tmp;//kelimedeki degistirilmis karakteri tekrar geri alabilmek icin kullanilmistir

int i,j,k,l;//index degiskenleri

int key;//hash fonksiyonunun uretecegi key degerini tutar

int h1,h2;//hashing fonksiyonlarinin degerlerini tutarlar

int MM,M;//hash tablosunun uzunlugunu tutar

int found=0;//sonucun donduruldugu bayrak degiskeni

int x=0;//Eger sozlukte hic onerilecek kelime yoksa bunu kullaniciya belirtmek icin //kullanilmistir

M=487;//load factor=0.7 kabul edildiginde tablo uzunlugu en kucuk asal sayi 487 olur

MM=M-1;

for(i=0;i<strlen(kelime);i++)

{

tmp=kelime[i];

for(j=97;j<=122;j++)//a'nin ascii karsiligi=97 ,z'nin ascii karsiligi=122

{

kelime[i]=j;

k=0;

found=0;

key=0;

for(l=0;l<strlen(kelime);l++)

{

key+=(kelime[l]-'a')\*26\*l;

}

while(found==0)

{

h1=key%M;

h2=1+(key%MM);

if(strcmp(hashTable[(h1+k\*h2)%M],kelime)==0)

found=1;

else if(strcmp(hashTable[(h1+k\*h2)%M],"#")==0)

found=-1;

k++;

}

if(found==1)

{

printf("%s ",kelime);

x=1;

}

}

kelime[i]=tmp;

}

if(x==0)

printf(" ----------\nSozlukte onerilebilecek kelime yok!!\n");

}

int main()

{

char \*\*dictionary;//sozlukte yer alan kelimeleri tutar

char \*\*hashTable;//hash tablosundaki kelimeleri tutar

char kelime[50];//kullanicidan alinan kelimeyi tutar

int i;//index degiskeni

dictionary=(char \*\*)malloc(sizeof(char \*) \* 500);

for (i = 0; i < 500; ++i)

dictionary[i] = (char \*)malloc(sizeof(char) \* 50);

hashTable=(char \*\*)malloc(sizeof(char \*) \* 500);

for (i = 0; i < 500; ++i)

hashTable[i] = (char \*)malloc(sizeof(char) \* 50);

createHashTable(hashTable,dictionary);

readHashTable(hashTable);

printf("string=");

gets(kelime);

search(kelime,hashTable);

free(hashTable);

free(dictionary);

return 0;

}