

Algoritma Analizi

Dersi 2.Ödev

Raporu

Dynamic Programming ve Hashing Algoritması Uygulaması

Bu uygulamada verilen sözlükteki kelimeler Hashing yöntemiyle saklanmıştır. Test olarak verilen kelimelerden yazım hatalı olanlara Levenshtein Distance mesafesi ile benzer doğru kelimeler bulunup yazım düzeltme önerisi detaylı olarak gerçekleştirilmiştir.

M.Yasin SAĞLAM

15011804

ALGORİTMA ANALİZİ

GRUP 1

Algoritma Analizi Dersi 2.Ödev Raporu

Dynamic Programming ve Hashing Algoritması Uygulaması

Yöntem

Verilen problemde bir sözlük ve test kelimeleri verilmiştir. Sözlükteki kelimeler hashing uygulanarak saklanacak ve test kelimelerinin içerisinde olan kelimelerin hashing uygulanarak sözlük içerisinde aranması işlemi ilk olarak beklenmektedir. Eğer kelime sözlükte yok ise bu kelimeye en benzer olan kelime edit distance yöntemiyle Levenshtein mesafesi kullanılarak bulunacak ve bulunan sözlük kelimesinin nasıl önerildiği (yani kelimenin diğer kelimeye hangi operatörler uygulanarak dönüştüğü) bilgisinin bir dosyada saklanması ve efektif olarak kodlanması beklenmektedir. Efektif olarak kodlayabilmek için rekürsif kodlama yerine Dinamik Programlama kullanılarak çözüme ulaşmak gerekmektedir. Öncelikle istenilen hashing algoritması **cal hash** adlı fonksiyonda kelimelerin soruda verilen hashing yöntemine göre bir önceden allocate edilmiş bir hash tablosuna atanması işlemini gerçekleştirmektedir. Kod karmaşasını azaltmak için oluşturulan hash tablosu **save hash** fonksiyonu ile dosyaya yazılmış, **load hash** fonksiyonu ile de dosyadan okunmuştur. Bunun yanında hash tablosunda kelime arama işlemini gerçekleştiren **look hash** fonksiyonu verilen kelimenin hash değerini hesaplayarak hash tablosunda kelimeyi aramaktadır. Bulamaması durumu ise Null gelmesi yada başlangıç noktasına geri dönülmesi olarak belirlenmiştir. Sonrasında ise yazılan **distance** adlı fonksiyonunda verilen 2 adet kelimenin edit distance ı Levenshtein Mesafesi uygulanarak Dynamic Programming yöntemiyle hesaplanmaktadır. Her bir hesaplama işleminden sonra hesaplama öncesi allocate edilen matris free yapılarak(yer bakımından efektif olması için) hesaplanan edit distance değeri dışarı döndürülmektedir. Verilen **save changes** adlı fonksiyonda, test kelimesine en yakın kelimenin (minimum edit distance a sahip kelime) distance matrisi yeniden oluşturulur ve kelimelerin birbirlerine benzerliklerinin nasıl uygulandığını gösteren backtracking algoritması uygulanarak ilgili değişikler fonksiyona verilen file pointer yardımıyla dosyaya yazılır. Yukarıda verilen fonksiyonlar main fonksiyonda menü dizayn edilerek çağrılmıştır. Kullanıcı 1 numaralı seçiminde hashlist oluşturmakta ve kaydetmektedir. 2 numaralı seçimde ise önceden kaydedilmiş hashlistesi varsa bunu dosyadan okumaktadır. 3 numaralı seçimde ise konsoldan girilen bir kelimenin varlığı önce hashlist üzerinden kontrol edilmekte, yoksa en benzer kelime ekrana basılmaktadır. 4 numaralı seçimde ise verilen test dosyasındaki kelimeler için 3 numaralı seçimde verilen işlemler yapılmakta ve değişimler problemde verilen örnekteki formatta dosyaya yazılmaktadır. Sözlükte bulunmamayı belirleyen threshold değeri ise batch0 test dosyasındaki test kelimelerinin maksimum edit distance değeri(6) + 2= 8 olarak seçilmiştir.

Uygulama

Örnek Sonuçlar

1-)Hashing(Konsol çalışması ve boşluklu yapıdaki txt dosya görseli)

```
47339 electoral
47340 electorate
47341 electorate(1)
47342 electorate's
47343 electorates
47344 electors
47345 electra
47346 electric
47347 denationalization
47348 electric's
47349 electrical
47350 electrically
47351 electrically(1)
47352 electricals
47353 electricar
47354 electrician
47355 electricians
```

```
Seç C:\Users\COMPUTER SCIENCE\Desktop\Dersler\Algoritma Analizi\Odev2\15011804.exe
sum = 2398 Hval = 62348 word = deinstitutionalization
sum = 1960 Hval = 50960 word = deinstitutionalize
sum = 1589 Hval = 41314 word = demographically
sum = 1821 Hval = 47346 word = denationalization
sum = 1936 Hval = 50336 word = denationalizations
sum = 1600 Hval = 41600 word = denationalizing
sum = 1728 Hval = 44928 word = dendrochronology
sum = 1713 Hval = 44538 word = denuclearization
sum = 1720 Hval = 44720 word = deoxyribonucleic
sum = 1739 Hval = 45214 word = departmentalize(1)
sum = 1709 Hval = 44434 word = departmentalized
sum = 1839 Hval = 47814 word = departmentalized(1)
sum = 1724 Hval = 44824 word = departmentalizes
sum = 1826 Hval = 47476 word = departmentalizing
*****
```

```
C:\Users\COMPUTER SCIENCE\CLionProjects\untitled2\hash.txt - Notepad++
Dosya Düzenle Ara Görünüm Kodlama Diller Ayarlar Araçlar Makrolar Çalıştır Eklentiler Pencerele 2
new 1 new 2 test.txt hash.txt
7721
7722
7723 a.'s
7724 ace
7725 add
7726 aga
7727 cea
7728 dad
7729 ebb
7730 fab
7731
7732
7733
7734
7735
7736
7737
7738
7739
7740
7741
7742
7743
7744
7745
7746
7747
7748
7749 aah
7750 ade
7751 aha
7752 b.'s
7753 bag
7754 bec
7755 cdc
7756 dae
7757 dea
7758 eda
7759 fac
7760 gab
7761
7762
7763
7764
7765
```


Sonuç

Open Adressing-Linear Probing hashing yapılmıştır. Bunun karmaşıklığı **load faktöre** bağlıdır. Bizim load faktör değerimiz soruda verilen kelimesayısı*2 değerine **en yakın asal sayıyı(m)** ifadesine uygun olarak yaklaşık olarak $\frac{1}{2}$ seçilmiştir. Aşağıda verilen karmaşıklık bağıntısına göre karmaşıklığımız başarılı bir aramada ortalama olarak $\frac{3}{2}$ başarılı olmayan bir aramada ise ortalama olarak $\frac{5}{2}$ değerini vermektedir.

Average RT – in a non-full hash table, assuming no previous removals, the average running time of insert, find, remove is

$$RT_{average}(n) = \begin{cases} \frac{1}{2} \left[1 + \frac{1}{1-\alpha} \right], & \text{for successful search} \\ \frac{1}{2} \left[1 + \frac{1}{(1-\alpha)^2} \right], & \text{for unsuccessful search} \end{cases}$$

Edit Distance karmaşıklığı ise dinamik programlama kullanıldığı için ortalama olarak $k_1 * k_2$ olmaktadır. Burada k_1 ve k_2 kelimelerin harf sayıları çarpımlarıdır. Biz bu işlemi normalde $N * N$ defa yapacak iken hashing sayesinde kötü bir durumda ortalama $N * (\frac{5}{2}) * k_1 * k_2$ olarak iyi bir durumda ise ortalama $N * (\frac{3}{2}) * k_1 * k_2$ karmaşıklığa indirgemiş olmaktadır. Bu değerleri toplam olarak ifade edecek olursak tüm kelimeler sözlükteyse best case $N * (\frac{3}{2})$ karmaşıklık yani kabaca N karmaşıklık eğer tüm kelimeler yazım hatalıysa lineer bir arama ve distance hesaplama olacağından toplam karmaşıklık $(N * N * k_1 * k_2 + N * (\frac{5}{2}))$ yani kabaca $N * N$ olacaktır.

LEVENSHTEİN RECURRENCE BAĞINTISI GÖRSELİ

- Initialization

$$D(i, 0) = i$$

$$D(0, j) = j$$

- Recurrence Relation:

For each $i = 1 \dots M$

For each $j = 1 \dots N$

$$D(i, j) = \min \begin{cases} D(i-1, j) + 1 \\ D(i, j-1) + 1 \\ D(i-1, j-1) + \begin{cases} 2; & \text{if } X(i) \neq Y(j) \\ 0; & \text{if } X(i) = Y(j) \end{cases} \end{cases}$$

- Termination:

$D(N, M)$ is distance

Kod

```

1.  /**
2.  @file
3.
4.  Bu uygulamada verilen sozlukteki kelimeler Hashing yontemiyle bir listede saklanmistir.
5.  Test olarak verilen kelimelerden yazim hatali olanlara Levenshtein Distance mesafesi ile benze
   r dođru kelimeler bulunup yazým düzeltme önerisi detayli olarak gerçekleştirilmiştir.
6.  Sonuclar ise dosyaya kaydedilmiştir.
7.  @author
8.
9.  Name           :      Muhammed Yasin SAGLAM
10. Student No      :      15011804
11. Date           :      28/10/2017
12. E-Mail         :      myasinsaglam1907@gmail.com
13. Compiler Used   :      GCC
14. IDE            :      DEV-C++(Version 5.11)
15. Operating System :      Windows 10 Educational Edition
16. */
17.
18. #include <stdio.h>
19. #include <stdlib.h>
20. #include <string.h>
21.
22. #define hashFile "hash.txt" //hash dosya adini tutan makro
23. #define dictFile "newdict.txt"//sozluk dosya adini tutan makro
24. #define testFile "test.txt"//test kelimelerinin dosya adini tutan makro
25. #define resultFile "result.txt"//sonuclarin dosya adini tutan makro
26. #define test_size 100 //test list size
27. #define size 133798 //dictionary size
28. #define hash_size 267601 //hash size
29. #define word_len 35 //max word length
30. #define threshold 7 //an edit distance threshold value to determine it not has a similiar word
   NONE word.
31. int distance(char *word1,char *word2);//iki kelimenin edit distance ini hesaplayan ve degeri d
   onduren fonksiyon
32. int cal_hash(char ***hashtable,char *word);//verilen kelimeyi hashtable da ilgili indise yazan
   fonksiyon
33. int save_hash(char ***hashtable);//oluřturulan hash listesini dosyaya kaydeden fonksiyon
34. void load_hash(char ***hashtable);//kayitli hash listesini dosyadan okuyan fonksiyon
35. int look_hash(char ***hashtable,char *word);//verilen kelimeyi hashtable da arayan bulursa 1 d
   onduren fonksiyon
36. int save_changes(FILE **fp,char *word1,char *word2);//2 kelimenin donusumunu dosyaya yazan fon
   ksiyon
37. int find_mindex(int *arr,int arr_size);//verilen dizinin en küçük elemaninin indisini donduren
   fonksiyon
38.
39. int main(){
40.     FILE *fp,*fpctest,*fpresult; //dosya okuma ve yazma islemleri icin file pointerlar tanimlan
   iyor
41.     int i,j,choice; //cevrim degiskenleri ve menu secimi degiskeni
42.     int min_ed,mindex,dist; //fonksiyonlardan donen edit distance degerlerini ve minimum bulma
   k icin indislerini tutacak degiskenler
43.     char **tests,**words,**hashtable; //test,sozluk ve hashtable icin pointerlar tanimlaniyor
44.
45.     words=(char**)calloc(size,sizeof(char*)); //sozlukteki kelimeler icin memory allocation
46.     for(i=0;i<size;i++){
47.         words[i]=(char*)calloc(word_len,sizeof(char));

```

```

48.     }
49.     fp=fopen(dictFile,"r"); //sozluk dosyasi aciliyor
50.     if(!fp){
51.         printf("Dictionary file not opened !!! Quitting...\n");
52.         exit(0);
53.     }
54.     i=0;
55.     while(!feof(fp)){
56.         fscanf(fp,"%s",words[i]); //kelimeler bellege aliniyor
57.         words[i]=strlwr(words[i]); //kelimeler kucuk harfe cevriyor
58.         i++;
59.     }
60.     fclose(fp);
61.     printf("\nDictionary read from file...\n");
62.     system("PAUSE");
63.     system("CLS");
64.     //hashtable dizisi olusturuluyor
65.     hashtable=(char**)calloc(hash_size,sizeof(char*)); //Hash tablosu icin memory allocation
66.     for(i=0;i<hash_size;i++){
67.         hashtable[i]=(char*)calloc(word_len,sizeof(char));
68.     }
69.
70.     //secenek menu
71.     printf("\n1.Create HashList and Save\n2.Load Hash List\n3.Test for only one input\n4.Write
results to file\n\nPlease enter the choice (0 for exit): ");
72.     scanf("%d",&choice); //Reading choice from the user
73.     system("CLS");
74.     while(choice!=0){
75.         if(choice==1){ //hash listesi olusturur ve dosyaya kaydeder
76.             system("CLS");
77.             printf("\nDictionary is hashing and saving as .txt file.\nWorking...\n");
78.             for(i=0;i<size;i++){
79.                 cal_hash(&hashtable,words[i]); //cal_hash cagrilir ve hash tablosu olusturulur
80.             }
81.             save_hash(&hashtable); //hashtable dosyaya yazilir
82.             system("CLS");
83.             printf("\nDictionary hashed and saved succesfully...\n");
84.             system("PAUSE");
85.         }
86.         if(choice==2){ //onceden olusturulmus hashlistesini dosyadan okur.
87.             system("CLS");
88.             load_hash(&hashtable); //hash tablosu dosyadan okunur
89.             printf("\nHash list read from file succesfully...\n");
90.             system("PAUSE");
91.         }
92.         if(choice==3){ //kullanci tarafından verilen 1 adet kelimeyi hash tablosunda arar bula
mazsa en yakin olan kelimeyi ekrana yazdirir.
93.             system("CLS");
94.             char *testword=(char*)calloc(word_len,sizeof(char));
95.             printf("Enter input to test hash functionality : ");
96.             scanf("%s",testword);
97.             testword=strlwr(testword);
98.             if(look_hash(&hashtable,testword)){
99.                 //system("CLS");
100.                 printf("\nFound in hashlist...\n");
101.             }
102.             else{
103.                 printf("\nGiven word is not found in hashlist...Searching similar word.
..\n");
104.                 min_ed=distance(words[0],testword);

```

```

105.         for(i=0;i<size;i++){
106.             if(distance(words[i],testword)<min_ed){
107.                 min_ed=distance(words[i],testword);
108.                 mindex=i;
109.             }
110.         }
111.         //system("CLS");
112.         if(min_ed>=threshold){ //kelimeye en benzer kelime thresholddan fazlays
a anlamsiz girdi demektir
113.             printf("\nNONE\n");
114.         }
115.         else{
116.             printf("\nMost similar word is %s Edit distance is %d\n",words[mind
ex],min_ed);
117.         }
118.     }
119.     system("PAUSE");
120. }
121. if(choice==4){
122.     system("CLS");
123.     tests=(char**)calloc(test_size,sizeof(char*)); //test edilecek kelimeler ici
n memory allocation
124.     for(i=0;i<test_size;i++){
125.         tests[i]=(char*)calloc(word_len,sizeof(char));
126.     }
127.
128.     //test edilecek kelimelerin dosyadan tests dizisine okunuyor ve küçük harfe
cevriliyor
129.     fptest=fopen(testFile,"r");
130.     if(!fptest){
131.         printf("Test file not opened!!! Quitting...\n");
132.         exit(0);
133.     }
134.     for(i=0;i<test_size;i++){
135.         fscanf(fp,"%s",tests[i]); //test kelimeleri okunuyor
136.         tests[i]=strlwr(tests[i]); //hepsi küçük harfe cevriliyor
137.     }
138.     fclose(fptest); //dosya kapatiliyor
139.     printf("\nTest inputs read from file...\n");
140.     system("PAUSE");
141.     system("CLS");
142.     printf("Working...\n");
143.     fpresult=fopen(resultFile,"w+"); //sonuclarin yazilacagi dosya aciliyor
144.     if(!fpresult){
145.         printf("\nResult file is not created!!! Quitting...\n");
146.         exit(0);
147.     }
148.     i=0;
149.     while(i<test_size){ //eger
150.         min_ed=100;
151.         if(look_hash(&hashtable,tests[i])){ //kelime sozlukte var mi hashlistte
n bak
152.             fprintf(fpresult,"%n%-35s\tOK\n",tests[i]);
153.         }
154.         else{ //yoksa
155.             for(j=0;j<size;j++){ //minimum ed ye sahip olan kelimeyi bul
156.                 dist=distance(tests[i],words[j]);
157.                 if(dist<min_ed){
158.                     min_ed=dist;
159.                     mindex=j;
160.                 }

```



```

161.         }
162.         if(min_ed>=threshold){ //eger thresholdu gecememisse
163.             fprintf(fpresult,"\n%-
35s\tNONE\n",tests[i]); //anlamsiz yaz dosyaya
164.         }
165.         else{ //gecmisse
166.             fprintf(fpresult,"\n%-35s%-
35s\t",tests[i],words[mindex]); //benzerini dosyaya yaz
167.             save_changes(&fpresult,tests[i],words[mindex]); //degisiklikler
i hesaplayan fonksiyonu cagirarak satiri tamamla
168.         }
169.     }
170.     i++;
171. }
172. fclose(fpresult); //sonuc dosyasi kapatiliyor
173. for(i=0;i<test_size;i++){ //test kelimeleri free
174.     free(tests[i]);
175. }
176. free(tests);
177. system("CLS");
178. printf("\nResults saved file(results.txt) successfully...\n");
179. system("PAUSE");
180. }
181. system("CLS");
182. system("COLOR a");
183. printf("\n1.Create HashList and Save\n2.Load Hash List\n3.Test for only one inp
ut\n4.Write results to file\n\nPlease enter the choice (0 for exit): ");
184. scanf("%d",&choice);
185. }
186.
187. //PROGRAM SONU
188.
189. for(i=0;i<size;i++){ //sozluk free yapiliyor
190.     free(words[i]);
191. }
192. free(words);
193. for(i=0;i<hash_size;i++){ //hashtable free yapiliyor
194.     free(hashtable[i]);
195. }
196. free(hashtable);
197.
198. system("PAUSE");
199. return 0;
200. }
201.
202.
203. //verilen kelimeyi hashtable da arayan bulursa 1 donduren fonksiyon
204. int look_hash(char ***hashtable,char *word){
205.     int len = strlen(word); //kelime uzunlugu okunuyor
206.     int i; //cevrir degiskeni
207.     int hash,sum=0; //hash degeri ve harflerin ascii toplamlarini tutan sum degiskeni
208.
209.     for(i=0;i<len;i++){
210.         sum+=word[i]; //herbir harfin ascii degeri toplaniyor
211.     }
212.
213.     hash=(sum*26)%hash_size; //hash fonsiyonu uygulanarak hash degeri hesaplaniyor
214.
215.     i=0;
216.     while(strcmp((*hashtable)[hash],"")!=0 && i<=hash_size){ //hash de ki indiste elema
n oldukca ve cevrir hash size dan az oldukca

```

```

217.         if(strcmp((*hashtable)[hash],word)==0){ //aranan kelime sozlukte var mi
218.             return 1; //bulundu anlamina gelir 1 dondurur
219.         }
220.         else{
221.             hash=(hash+1)%hash_size; //yoksa null gorene kadar indis ilerler
222.         }
223.         i++; //listeden kac elemana bakildi
224.     }
225.     return 0; //bulunamadi anlaminda cikis demektir eleman sozlukte yok null a denk gel
mis
226. }
227.
228. //verilen kelimeyi hashtable da ilgili indise yazan fonksiyon
229. int cal_hash(char ***hashtable,char *word){
230.     int len = strlen(word); //kelime uzunlugu okunuyor
231.     int i; //cevrilm degiskeni
232.     int hash,sum=0; //hash degeri ve harflerin ascii toplamlarini tutan sum degiskeni
233.
234.     for(i=0;i<len;i++){
235.         sum+=word[i]; //herbir harfin ascii degeri toplaniyor
236.     }
237.
238.     hash=(sum*26)%hash_size; //hash fonsiyonu uygulanarak hash degeri hesaplaniyor
239.
240.     if(strcmp((*hashtable)[hash],"")==0){ //eger tablodaki goz bossa
241.         strcpy((*hashtable)[hash],word); //elemani yaz
242.     }
243.     else{ //degilse
244.         while(strcmp((*hashtable)[hash],"") != 0){//bos olana kadar
245.             hash=(hash+1)%hash_size; //daireysel bicimde ilerle
246.         }
247.         strcpy((*hashtable)[hash],word); //sonrasinda ise elemani yerlestir
248.     }
249.
250.     return 0;
251. }
252.
253. //olupdurulan hash listesini dosyaya kaydeden fonksiyon
254. int save_hash(char ***hashtable){
255.     FILE *fp;
256.     int i;
257.     fp=fopen(hashFile,"w");
258.     if(!fp){
259.         printf("\nError occured!!! Hash file not created to write...\n");
260.         exit(0);
261.     }
262.     for(i=0;i<hash_size;i++){
263.         fprintf(fp,"%s\n",(*hashtable)[i]);
264.     }
265.     printf("\nHash file created named hash.txt\n");
266.     fclose(fp);
267.     return 0;
268. }
269.
270. //kayitli hash listesini dosyadan okuyan fonksiyon
271. void load_hash(char ***hashtable){
272.     FILE *fp;
273.     int i;
274.     char *temp;
275.     fp=fopen(hashFile,"rb+"); //hash dosyasi okuma modunda aciliyor
276.     if(!fp){

```

```

277.         printf("\nError occured!!! Hash file not opened...\n");
278.         exit(0);
279.     }
280.     size_t len;
281.     for(i=0;i<hash_size;i++){ //bopluklu yapıdaki hash dosyasından okuma yaparken olup
        an karakter fazlalıyđý hatasýnýn giderildiđi kýsým
282.         temp=(char*)malloc(sizeof(char)*50);
283.         fgets(temp,50,fp);
284.         len=strlen(temp)-2; //fazlalık olan son 2 karakteri alma
285.         memcpy((*hashtable)[i],temp,len);
286.         free(temp);
287.     }
288.     fclose(fp);
289.
290. }
291.
292. //iki kelimenin edit distance ini hesaplayan fonksiyon
293. int distance(char *word1,char *word2){
294.     int len1=strlen(word1)+1; //kelime 1 uzunlugu
295.     int len2=strlen(word2)+1; //kelime 2 uzunlugu
296.     int i,j; //cevrim degiskenleri
297.     char c1,c2; //karakter kiyaslama için degiskenler
298.     int deletion,insertion,change,minimum,edit_distance; //ceza islemlerinin uygulanmas
        ini saglayan degiskenler
299.     //2 boyutlu dinamik distance matrisi tanımlaniyor
300.     int **matrix=(int**)calloc(len1,sizeof(int*));
301.     for(i=0;i<len1;i++){
302.         matrix[i]=(int*)calloc(len2,sizeof(int));
303.     }
304.     if(!matrix){
305.         printf("Allocation error!!! Quitting...");
306.         exit(0);
307.     }
308.     //matris baslangic degerleri ataniyor (initialization)
309.     for(i=0;i<len1;i++){
310.         matrix[i][0]=i;
311.     }
312.     for(i=0;i<len2;i++){
313.         matrix[0][i]=i;
314.     }
315.     //matris hesaplamaları
316.     for(i=1;i<len1;i++){
317.         c1=word1[i-1]; //ilk kelimenin karakteri
318.         for(j=1;j<len2;j++){
319.             c2=word2[j-1]; //ikinci kelimenin karakteri
320.             if(c1==c2){ //birbirlerine esitler mi?
321.                 matrix[i][j]=matrix[i-1][j-
                    1]; //esitse sol ustteki degeri ver copy olmus demektir ceza=0
322.             }
323.             else{//degilse
324.                 deletion = matrix[i-1][j] + 1; //silme olussa +1
325.                 insertion = matrix[i][j-1] + 1; //ekleme varsa +1
326.                 change = matrix[i-1][j-1] + 2; // degisim varsa ceza +2
327.                 //minimum bulunur
328.                 minimum = deletion;
329.                 if (insertion < minimum) {
330.                     minimum = insertion;
331.                 }
332.                 else if (change < minimum) {
333.                     minimum = change;
334.                 }
            }
        }
    }
}

```

```

335.         matrix[i][j] = minimum; //ilgili goze minimum deger atamasi yapilir
336.     }
337. }
338. }
339. edit_distance = matrix[len1-1][len2-1];
340. //free matrix
341. for(i=0;i<len1;i++){
342.     free(matrix[i]);
343. }
344. free(matrix);
345.
346. return edit_distance;
347. }
348.
349. //2 kelimenin donusumunu dosyaya yazan fonksiyon
350. int save_changes(FILE **fp,char *word1,char *word2){
351.     int len1=strlen(word1)+1; //kelime 1 uzunlugu
352.     int len2=strlen(word2)+1; //kelime 2 uzunlugu
353.     int i,j; //cevrim degiskenleri
354.     char c1,c2; //karakter kiyaslama icin degiskenler
355.     int deletion,insertion,change,minimum,edit_distance,minindex; //ceza islemlerinin uyg
        ulanmasini saglayan degiskenler
356.     int *way=(int*)calloc(3,sizeof(int)); //backtrack icin yon degerlerinin tutulacagi
        way adli dizi
357.     if(!way){
358.         printf("Allocation error!!! Quitting...");
359.         exit(0);
360.     }
361.     //2 boyutlu dinamik distance matrisi tanimlaniyor
362.     int **matrix=(int**)calloc(len1,sizeof(int*));
363.     for(i=0;i<len1;i++){
364.         matrix[i]=(int*)calloc(len2,sizeof(int));
365.     }
366.     if(!matrix){
367.         printf("Allocation error!!! Quitting...");
368.         exit(0);
369.     }
370.     //matris baslangic degerleri ataniyor (initialization)
371.     for(i=0;i<len1;i++){
372.         matrix[i][0]=i;
373.     }
374.     for(i=0;i<len2;i++){
375.         matrix[0][i]=i;
376.     }
377.     //matris hesaplamalari
378.     for(i=1;i<len1;i++){
379.         c1=word1[i-1]; //ilk kelimenin karakteri
380.         for(j=1;j<len2;j++){
381.             c2=word2[j-1]; //ikinci kelimenin karakteri
382.             if(c1==c2){ //birbirlerine esitler mi?
383.                 matrix[i][j]=matrix[i-1][j-1];
        //esitse sol ustteki degeri ver copy olmus demektir ceza=0
384.             }
385.             else{//degilse
386.                 deletion = matrix[i-1][j] + 1; //silme olmus +1
387.                 insertion = matrix[i][j-1] + 1; //ekleme varsa +1
388.                 change = matrix[i-1][j-1] + 2; // degisim varsa ceza +2
389.                 //minimum bulunur
390.                 minimum = deletion;
391.                 if (insertion < minimum) {
392.                     minimum = insertion;

```

```

393.         }
394.         else if (change < minimum) {
395.             minimum = change;
396.         }
397.         matrix[i][j] = minimum; //ilgili goze minimum deger atamasi yapilir
398.     }
399. }
400. }
401. edit_distance = matrix[len1-1][len2-1];
402.
403.
404. //backtracking
405. i=len1-1;
406. j=len2-1;
407.
408.
409.
410. while(i!=0 && j!=0){ //kenarlara gelene kadar
411.     way[0] = matrix[i-1][j]; //0.goz yukaridaki deger --delete
412.     way[1] = matrix[i][j-1]; //1.goz soldaki deger --insert
413.     way[2] = matrix[i-1][j-1]; //2.goz caprazdaki deger --change degerini saklar
414.     mindex=find_mindex(way,3); //degeri minimum olan yonu dondurur
415.     if(mindex == 0){ //eger yukaridan gelinmisse
416.         fprintf(*(fp)," (del->%c) ",word1[--i]);
417.         //i 1 azaltilir yani yukari ilerlenir ve test kelimesinden ilgili harf silinmeli
418.     }
419.     else if(mindex == 1){ //eger soldan gelinmisse
420.         fprintf(*(fp)," (ins->%c) ",word2[--j]);
421.         //j 1 azaltilir yani sola ilerlenir ve test kelimesine ilgili harf eklenir
422.     }
423.     else if(matrix[i-1][j-1] != matrix[i][j]){
424.         //eger caprazdaki deger degismisse
425.         fprintf(*(fp)," (chg->%c,%c) ",word1[--i],word2[--j]);
426.         //i ve j 1 azaltilir yani capraza ilerlenir ve degisiklik yapilir
427.     }
428.     else{ //eger capraz ayniyse ve minimumsa degisiklik yok devam edilir
429.         fprintf(*(fp)," %c ",word1[--i]);
430.         j--;
431.     }
432. }
433.
434. while (j>0){ //ustteki kenara gelinmisse surekli sola gidilir ve ekleme yapilir
435.     fprintf(*(fp), " (ins->%c) ", word2[--j]);
436. }
437.
438. fputs("\n",*(fp));
439.
440. //free matrix
441. for(i=0;i<len1;i++){
442.     free(matrix[i]);
443. }
444. free(matrix);
445.
446. return edit_distance;
447. }
448.

```

```
449. //verilen dizinin en küçük elemanın indisini döndüren fonksiyon
450. int find_mindex(int *arr,int arr_size){
451.     int i,min,mindex;
452.     min=arr[0];
453.     mindex=0;
454.     for(i=1;i<arr_size;i++){
455.         if(arr[i]<min){
456.             min=arr[i];
457.             mindex=i;
458.         }
459.     }
460.     return mindex;
461. }
```