YILDIZ TEKNİK ÜNİVERSİTESİ
ELEKTRİK-ELEKTRONİK FAKÜLTESİ
BİLGİSAYAR MÜHENDİSLİĞİ BÖLÜMÜ

2020-2021 GÜZ YARIYILI

BLM3021 ALGORİTMA ANALİZİ DERSİ

GRUP-2

3.ÖDEV 2.PROBLEM RAPORU

Konu: Dinamik Programlama

Hazırlayan: Mehmet Hayri Çakır – 16011023

Dersin Yürütücüsü: Dr. Öğr. Üyesi Mehmet Amaç GÜVENSAN

İstanbul, Aralık 2020

İçindekiler

1. Yöntem

1.1. Problem

Bu ödevde, sorgulanan bir cümlede yanlış yazılmış kelimeler varsa bu kelimelerin yerine doğru kelimeler öneren bir sistem tasarlanacaktır.

1.2. Çözüm

Öncelikle bir metin belgesindeki kelimeler hashing ile bir sözlüğe eklenecektir. Kullanıcı tarafından girilen bir cümledeki her bir kelime önce sözlükte, eğer orada bulunamazsa yanlış yazılmış kelimeler ve bu kelimelere karşılık gelen, kullanıcı tarafından daha önceden kabul görmüş kelimelerin olduğu sözlükte aranacak ve buna göre de Levenshtein Edit Distance algoritmasına göre mesafesi 1 olan, yoksa 2 olan kelimeler kullanıcıya önerilecektir. Eğer kullanıcı yapılan öneriler yerine kendi yazdığı kelimeyi tekrar girerse o kelime kullanıcının yazdığı şekliyle ekrana basılacaktır. Hash tabloları *openaddress* ile oluşturuldu. Çakışma problemini çözmek için ise *double hashing* yöntemi kullanıldı. Kod başında define kullanılarak; sözlük dosyası adı "smallDictionary.txt" maksimum kelime uzunluğu "32", maksimum cümle uzunluğu "64", R "2", M(hash tablolarının boyutu) "421" olarak belirlenmiştir. İhtiyaç duyulması durumunda değiştirilmeleri kullanıcı sorumluluğundadır.

1.2.1. Bonus Çözüm

Ödev dokümanındaki bonus problemin çözümü içinse, iki kelime karşılaştırılırken her matris hücresine yazma işleminden sonra *currentMinDistance* değişkeni güncellenir. Sonraki satıra geçtiğimizde eğer *currentMinDistance, maxDistance* değişkeninden büyükse, bir üst satırdaki en küçük distance bile, *maxDistance*'dan büyük demektir. Böylece, sol üst çapraz, sol veya üst hücreden alacağımız her değerin *maxDistance*'dan büyük olacağını anlarız ve döngüyü kırıp sözlükteki bir sonraki kelimeye geçeriz.

2. Uygulama

2.1. Fonksiyonlar

Main fonksiyonu hariç toplam 10 fonksiyon kullanılmıştır:

- `int hash1(int key)`
- `int hash2(int key)`
- `int doubleHash(int key, int i)`
- `int produceKey(char* word)`
- `int searchWord(char** dictionary, char* word)`
- `int searchWord2(WORD_COUPLE* wrong_words_dictionary, char* word)`
- `int editDistance(char** dictionary, char* word)`
- `void addToWrongWordsDictionary(WORD_COUPLE* wrong_words_dictionary, char* word, char* corrected_word)`
- `void getNewSentence(char** dictionary, WORD_COUPLE* wrong_words_dictionary)`
- `void populateDictionary(char** dictionary)`

2.1.1. int hash1(int key)

Birincil hash fonksiyonu. Gönderilen key karşılığında hash table'a erişmek için bir indis döndürür.

2.1.2. int hash1(int key)

İkinci hash fonksiyonu. Çakışma olması durumunda double hashing yöntemini kullandığım için yazılmıştır.

2.1.3. int doubleHash(int key, int i)

Çakışma olması durumunda bu fonksiyon kullanılır.

2.1.4. int produceKey(char* Word)

Horner metodunu kullanarak, parametre olarak gelen kelimeye karşılık bir key döndürür. R değeri 2 alınmıştır. İstenilmesi durumunda kodun başındaki define satırından değiştirilebilir.

2.1.5. int searchWord(char** dictionary, char* word)

Parametre olarak verilen kelimeyi, sözlükte aramak için kullanılır. Kelime bulunamazsa; sözlük doluysa veya boş hücreye denk geldiyse 0 döndürür.

2.1.6. int searchWord2(WORD_COUPLE* wrong_words_dictionary, char* word)

Parametre olarak verilen kelimeyi, yanlış kelimeler sözlüğünde aramak için kullanılır. Eğer kelime bulunursa, kelimenin yanlış kelimeler sözlüğündeki indisini döndürür. Eğer kelime bulunamazsa; sözlük doluysa M+1, boş hücreye denk geldiyse o hücrenin indisinin -1 ile çarpılmış halini döndürür. Bunun sebebi, yanlış kelimeler sözlüğüne yeni bir kelime ekleneceği zaman arama yapıldığında o kelime sözlükte bulunamazsa tek adımda indisi de almış olmaktır. Eğer bu şekilde bir çözüm yapmasaydım boş indisi bulabilmek için tekrardan bir fonksiyon yazmam gerekecekti.

2.1.7. int editDistance(char** dictionary, char* word)

Parametre olarak gelen kelimenin, sözlükteki her bir kelimeye olan mesafesini Levenshtein Edit Distance algoritmasını kullanarak bulur. Eğer mesafenin *maxDistance* değişkeninden (ödev dokümanında k = 2 olarak verildiği için kodda varsayılan değer olarak 2 atanmıştır.) fazla olacağı kesinleşirse, döngüden yarıda çıkıp sözlükteki sonraki kelimeye geçer. Tüm sözlük gezilip mesafeler hesaplandıktan sonra kullanıcıya 1 mesafede bulunan kelimeleri önerir. Eğer 1 mesafede bulunan kelime yoksa 2 mesafedeki kelimeleri önerir. 2 mesafede de kelime yoksa -2 döndürür ve kullanıcının girdiği kelime ekrana basılır. Eğer kullanıcı önerilen kelimeler arasından biri seçmek yerine kendi yazdığı kelimede ısrar ederse, -1 döndürür ve kullanıcının girdiği kelime ekrana basılır. Eğer kullanıcı önerilen kelimelerden birini seçerse, o kelimenin sözlükteki adresini döndürür.

2.1.8. void addToWrongWordsDictionary(WORD_COUPLE* wrong_words_dictionary, char* word, char* corrected_word)

Eğer kullanıcı kendisine önerilen kelimeler arasından birini seçerse; yanlış girilen kelimeyi ve kullanıcının önerilen kelimeler arasından seçtiği kelimeyi, yanlış kelimeler sözlüğüne ekler.

2.1.9. void getNewSentence(char** dictionary, WORD_COUPLE* wrong_words_dictionary)

Kullanıcıdan yeni bir cümle alıp bu cümlenin her bir kelimesi için;

1) Kelime, sözlük tablosunda aranır. Eğer kelime, sözlük tablosunda
   a) Varsa kelime doğrudur. O kelime için işlem tamamlanır.
   b) Yoksa, hatalı kelimeler tablosunda aranır.
2) Kelime, hatalı kelimeler tablosunda
   a) Yoksa, *editDistance* fonksiyonu çağrılır. *editDistance* fonksiyonu sonlanınca, duruma göre *addToWrongWordsDictionary* fonksiyonu çağrılabilir.
   b) Varsa, hatalı kelime tablosunda bu hatalı kelime bulunur ve bu kelime için daha önce önerilip kullanıcı tarafından kabul edilmiş olan doğru kelime kullanıcıya önerilir.

2.1.10. void populateDictionary(char** dictionary)

Sözlük dosyasındaki kelimelerle, sözlük tablosunu doldurur. Sözlük dosyasının varsayılan adı *smallDictionary.txt* olarak kodun başında define ile belirlenmiştir.

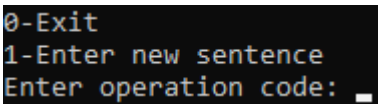
2.1.11. int main()

Program ilk çalıştığında hash tablosuna yer açar. *populateDictionary* fonksiyonunu çağırarak sözlük tablosunun içini doldurur. Sonrasında switch-case yapısı ile kullanıcıdan alınan girdilere karşılık gelen fonksiyonlar çağrılır. (0-Programı kapatır, 1-Kullanıcının yeni cümle girmesini sağlar.)

2.2. Ekran Çıktıları

Bu bölümde 4 fonksiyonun (*updateHashTable, searchWord, printTable, maxStepsToFindAWord*) kullanımı sırasında ortaya çıkan farklı durumları gösteren ekran çıktıları eklenmiştir. Yazdırma esnasında kolay okunabilmesi için formatlama parametreleri kullanılması sonucu bazı kısımlarda kelimelerden sonra ekstra boşluklar oluşmuştur.

2.2.1. Program Menüsü



2.2.2. Bütün Kelimelerin Sözlükte Bulunması Durumu

Girilen bütün kelimeler sözlükte mevcutsa, cümle değiştirilmeden ekrana basılır.

```
0-Exit
1-Enter new sentence
Enter operation code: 1
Enter the sentence without special characters: it is hold

New version of given sentence: 'it is hold'

0-Exit
1-Enter new sentence
Enter operation code:
```

2.2.3. Bir Kelimenin sözlükte Bulunmaması Durumu

2.2.3.1. Hatalı kelimeler sözlüğünde de bulunmaması durumu

Eğer girilen kelime hatalı kelime sözlüğünde de yoksa, sözlükteki kelimelere olan mesafesi hesaplanıp kullanıcıya öneriler yapılır. Kullanıcı burada "*good"* kelimesini seçtiği için, tekrar aynı hatalı kelime("*gold*") girilmesi durumunda kullanıcıya hatalı kelime tablosundan "*good"* kelimesi önerilecektir.

```
0-Exit
1-Enter new sentence
Enter operation code: 1
Enter the sentence without special characters: it is gold
'gold' is not in the dictionary. Did you mean: 'hold' 'good'
Enter the chosen word: good

New version of given sentence: 'it is good'

0-Exit
1-Enter new sentence
Enter operation code:
```

2.2.3.2. Hatalı kelimeler sözlüğünde bulunması durumu

```
0-Exit
1-Enter new sentence
Enter operation code: 1
Enter the sentence without special characters: it is gold
'gold' is not in the dictionary. Did you mean: 'hold' 'good'
Enter the chosen word: good

New version of given sentence: 'it is good'

0-Exit
1-Enter new sentence
Enter operation code: 1
Enter the sentence without special characters: it is gold
'gold' is not in the dictionary. Did you mean: 'good'
Enter the chosen word: good

New version of given sentence: 'it is good'

0-Exit
1-Enter new sentence
Enter operation code:
```

2.2.3.3. Hem hatalı kelimeler sözlüğünde bulunmayıp hem de sözlükte maksimum 2 mesafede kelime bulunmaması durumu

Böyle bir durumda, kullanıcının girdiği kelime doğru kabul edilip ekrana basılır.

```
0-Exit
1-Enter new sentence
Enter operation code: 1
Enter the sentence without special characters: it is perfect
'perfect' is not in the dictionary. Couldn't find similar word in dictionary too.

New version of given sentence: 'it is perfect'

0-Exit
1-Enter new sentence
Enter operation code:
```

2.2.3.4. Kullanıcının önerilen kelimeler yerine kendi kelimesini girmesi

Böyle bir durumda, kullanıcının girdiği kelime doğru kabul edilip ekrana basılır.

```
0-Exit
1-Enter new sentence
Enter operation code: 1
Enter the sentence without special characters: it is gold
'gold' is not in the dictionary. Did you mean: 'hold' 'good'
Enter the chosen word: gold

New version of given sentence: 'it is gold'

0-Exit
1-Enter new sentence
Enter operation code:
```

3. Kod

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

#define DICTIONARY_FILE_NAME "smallDictionary.txt"
#define MAX_WORD_LENGTH 32
#define MAX_SENTENCE_LENGTH 64
#define R 2
#define M 421
#define min(X, Y) (((X) < (Y)) ? (X) : (Y))

typedef struct
{
	char* wrong_word;
	char* correct_word;
}WORD_COUPLE;

int hash1(int key);
int hash2(int key);
int doubleHash(int key, int i);
int produceKey(char* word);
int searchWord(char** dictionary, char* word);
int searchWord2(WORD_COUPLE* wrong_words_dictionary, char* word);
int editDistance(char** dictionary, char* word);
void addToWrongWordsDictionary(WORD_COUPLE* wrong_words_dictionary, char* word, char*
corrected_word);
void getNewSentence(char** dictionary, WORD_COUPLE* wrong_words_dictionary);
void populateDictionary(char** dictionary);
```

```
//birinci hash fonksiyonu
int hash1(int key)
{
	key = key % M;
	if (key < 0)
	{
		key += M;
	}
	return key;
}

//ikinci hash fonksiyonu
int hash2(int key)
{
	return 1 + key % (M - 1);
}

//double hash icin kullanilan fonksiyon
int doubleHash(int key, int i)
{
	key = ((key % M) + i * hash2(key)) % M;
	if (key < 0)
	{
		key += M;
	}
	return key;
}

//horner metodu ile key ureten fonksiyon. odevde buyuk kucuk harf duyarsiz dedigi icin gelen her
kelime once kucuk harflere donusturulup oyle saklanir.
int produceKey(char* word)
{
	int key = 0, i;
	for (i = 0; i < strlen(word); i++)
	{
		key += (pow(R, i) * (tolower(word[strlen(word) - 1 - i]) - 'a' + 1));
	}
	return key;
}

//sozlukte bir kelime aramak icin kullanilir.
//kelime bulunamazsa ve bos hucreye denk geldiysek veya sozluk doluysa, -1 doner.
//kelime bulunursa 0 doner. indise ihtiyacimiz yok cunku kelime dogru yazilmistir.
int searchWord(char** dictionary, char* word)
{
	int i = 0; //dongu degiskeni
	int j = 0; //dongu degiskeni
	int key = produceKey(word); //kelimeye karsilik gelen key degiskeni
	int address = hash1(key); //sozlugun indisini belirten degisken

	//programda buyuk kucuk harf duyarsiz kabul ettigim icin kelime kucuk harfe donusturulur.
	for (i = 0; i < strlen(word); i++)
	{
		word[i] = tolower(word[i]);
	}

	//sozlugun o hucresi bos mu?
	if (dictionary[address] == NULL)
	{
		//bossa kelime sozlukte yoktur
		return -1;
	}
	//bos degilse aradigimiz kelime o hucredeki kelime mi?
	else if (strcmp(dictionary[address], word) == 0)
	{
```

```
			//cevap evetse kelimeyi bulduk. kelime dogru yazilmis.
			return 0;
		}
		else
		{
			//cevap hayirsa double hash yontemine basvurup o kelimeyi ararız.
			i = 1;
			j = 0;
			//o kelimeyi bulana veya bos bir hucreye denk gelene kadar devam ederiz
			while ((dictionary[address] != NULL) && (strcmp(dictionary[address], word) != 0))
			{
				address = doubleHash(key, i);
				i++;
				if (i == M)
				{
					//i sozluk boyutunu asarsa kelime sozlukte yoktur. (sozlugun dolu oldugu
durum icin sonsuz donguden kacinma amacli yazdim)
					return -1;
				}
			}
			//hucredeki kelime aradigimiz kelime mi?
			if (dictionary[address] == NULL)
			{
				//bos hucreye denk geldik. kelime sozlukte yoktur.
				return -1;
			}
			else if (strcmp(dictionary[address], word) == 0)
			{
				//cevap evetse aradigimiz kelimeyi bulduk, kelime dogru yazilmis.
				return 0;
			}
		}
}

//parametre olarak gelen kelimeyi, yanlis kelimeler sozlugunde arar.
//bulmasi durumunda yanlis kelimenin karsiligi olarak tabloda bulunan WORD_COUPLE structinin
tablodaki indisini
//bulamamasi durumundaysa farkli bir deger dondurur. tablo doluysa M+1 doner. degil ise denk gelinen
bos gozun adresini doner.
//tabloya yeni kelime eklenecekse bu adres lazim olacak.
int searchWord2(WORD_COUPLE* wrong_words_dictionary, char* word)
{
	int i = 0; //dongu degiskeni
	int j = 0; //dongu degiskeni
	int key = produceKey(word); //kelimeye karsilik gelen key degiskeni
	int address = hash1(key); //sozlugun indisini belirten degisken

	//programda buyuk kucuk harf duyarsiz kabul ettigim icin kelime kucuk harfe donusturulur.
	for (i = 0; i < strlen(word); i++)
	{
		word[i] = tolower(word[i]);
	}

	//sozlugun o hucresi bos mu?
	if (wrong_words_dictionary[address].wrong_word == NULL)
	{
		//bos hucreye denk geldik. kelime sozlukte yoktur. sozluge kelime eklerken bos indisi
bilmek gerektigi icin yine de adresi donduruyorum,
		//sonrasinda bu fonksiyonun cagrildigi yerde o adres bos mu diye bakacagim.
		return address * -1;
	}
	//bos degilse aradigimiz kelime o hucredeki kelime mi?
	else if (strcmp(wrong_words_dictionary[address].wrong_word, word) == 0)
	{
		//cevap evetse kelimeyi bulduk. onerilen kelimeyi kullaniciya oner.
		return address;
```

```
	}
	else
	{
		//cevap hayirsa double hash yontemine basvurup o kelimeyi arariz.
		i = 1;
		j = 0;
		//o kelimeyi bulana veya bos bir hucreye denk gelene kadar devam ederiz
		while ((wrong_words_dictionary[address].wrong_word != NULL) &&
(strcmp(wrong_words_dictionary[address].wrong_word, word) != 0))
		{
			address = doubleHash(key, i);
			i++;
			if (i == M)
			{
				//i sozluk boyutunu asarsa kelime sozlukte yoktur. (sozlugun dolu oldugu
durum icin sonsuz donguden kacinma amacli yazdim)
				return M + 1;
			}
		}
		//hucredeki kelime aradigimiz kelime mi?
		if (wrong_words_dictionary[address].wrong_word != NULL)
		{
			//bos hucreye denk geldik. kelime sozlukte yoktur. sozluge kelime eklerken bos
indisi bilmek gerektigi icin yine de adresi donduruyorum,
			//sonrasinda bu fonksiyonun cagrildigi yerde o adres bos mu diye bakacagim.
			return address * -1;
		}
		else if (strcmp(wrong_words_dictionary[address].wrong_word, word) == 0)
		{
			//cevap evetse aradigimiz kelimeyi bulduk, onerilen kelimeyi kullaniciya oner.
			return address;
		}
	}
}

//parametre olarak gelen kelimenin, sozlukteki her bir kelimeye olan mesafesi Levenshtein Edit
Distance algoritmasiyla bulunur.
//eger mesafenin maxDistance(odev dokumaninda k = 2 olarak verilmis) degiskeninden fazla olacagi
kesinlesirse, o kelimeyi atlar ve sonrakine gecer.
//eger kullanici onerilen kelimelerden birini secerse o kelimenin sozlukteki adresi return edilir.
//eger kullanici kendi yazdigi kelimede israr edip onerilenlerden birini secmezse -1 dondurulur.
//eger maxDistance kadar uzaklikta bir kelime bulunamazsa -2 dondurulur ve kullanicinin yazdigi
kelime ekrana basilir.
int editDistance(char** dictionary, char* word)
{
	int i;//dongu degiskeni
	int j;//dongu degiskeni
	int k;//dongu degiskeni
	int currentMinDistance;//iki kelime arasindaki suanki minimum distance.
	int maxDistance = 2;//kabul edilebilir maksimum distance, odevde k = 2 olarak verilen deger.
	int proceedToNextWordInDictionary = 0;//eger currentMinDistance > maxDistance sarti
saglanirsa donguyu kirmak icin kullanilan flag
	int oneDistanceExist = 0;//girilen kelimeye mesafesi 1 olan, sozlukteki kelime sayisi
	int twoDistanceExist = 0;//girilen kelimeye mesafesi 2 olan, sozlukteki kelime sayisi
	int** distanceMatrix;//iki kelime arasindaki mesafe matrisi
	int* toSuggestWordIndexes1Distance;//1 mesafeli kelimelerin sozlukteki adreslerini tutan dizi
	int* toSuggestWordIndexes2Distance;//2 mesafeli kelimelerin sozlukteki adreslerini tutan dizi
	char input_word[MAX_WORD_LENGTH];//onerilen kelimeler sonrasinda kullanicidan bir kelime
alinir.

	toSuggestWordIndexes1Distance = (int*)calloc(1, sizeof(int));
	toSuggestWordIndexes2Distance = (int*)calloc(1, sizeof(int));
	distanceMatrix = (int**)calloc(strlen(word) + 1, sizeof(int*));

	for (i = 0; i < strlen(word) + 1; i++)
	{
```

```
		distanceMatrix[i] = (int*)calloc(1, sizeof(int));
	}

	//butun sozluk boyunca dongude kaliriz
	for (i = 0; i < M; i++)
	{
		proceedToNextWordInDictionary = 0;
		//sozlugun i. indisinde bir kelime var mi?
		if (dictionary[i] != NULL)
		{
			//eger varsa, matrisin satirlari icin i. indisteki kelime uzunlugu + 1 kadar
yer acilir.
			for (j = 0; j < strlen(word) + 1; j++)
			{
				distanceMatrix[j] = (int*)realloc(distanceMatrix[j],
(strlen(dictionary[i]) + 1) * sizeof(int));
				//ilk satir degerleri atanir
				distanceMatrix[j][0] = j;

			}
			for (j = 0; j < strlen(dictionary[i]) + 1; j++)
			{
				//ilk sutun degerleri atanir.
				distanceMatrix[0][j] = j;
			}

			//matrisin kalan hucrelerinin doldurulduğu dongu.
			for (j = 1; j < strlen(word) + 1; j++)
			{
				//o ana kadarki minimum distance takibini yapmak icin.
				currentMinDistance = maxDistance + 1;
				for (k = 1; k < strlen(dictionary[i]) + 1; k++)
				{
					//kelimelerin o harfleri ayni mi?
					if (word[j - 1] == dictionary[i][k - 1])
					{
						//ayniysa sol ust caprazdaki degeri aynen tasiriz. copy
islemi oldugu icin cost eklenmez.
						distanceMatrix[j][k] = distanceMatrix[j - 1][k - 1];
					}
					else
					{
						//ayni degilse insert veya delete olmus olabilir. sol ust
capraz, ust ve sol gozdeki degerlerden
						//en kucuk olani alip cost(1) ekleriz.
						distanceMatrix[j][k] = min(min(distanceMatrix[j - 1][k -
1], distanceMatrix[j - 1][k]), distanceMatrix[j][k - 1]) + 1;
					}
					//eger matrisin o hucresindeki distance currentMinDistance'dan
kucukse yeni current min o hucredeki deger olur.
					currentMinDistance = min(currentMinDistance,
distanceMatrix[j][k]);
				}
				//eger current min, max'i gectiyse daha fazla devam etmeye gerek yoktur.
sozlukteki sonraki kelimeye bakariz.
				if (currentMinDistance > maxDistance)
				{
					//donguyu kirmak icin kullanilan flag 1 olur.
					proceedToNextWordInDictionary = 1;
					break;
				}
			}
			//eger bu flag 1 olduysa distance kesinlikle maxDistancedan buyuk olacaktir.
			//sonraki kelimeye devam ederiz.
			if (proceedToNextWordInDictionary)
			{
```

```
                        continue;
                }
                //eger matris sonuna ulasinca iki kelime arasindaki mesafe 1 ise
                if (distanceMatrix[strlen(word)][strlen(dictionary[i])] == 1)
                {
                        //ilgili kelimenin sozlukteki indisini, 1 mesafeli kelime onerilerinin
oldugu diziye ekleriz.
                        toSuggestWordIndexes1Distance =
(int*)realloc(toSuggestWordIndexes1Distance, (oneDistanceExist + 1) * sizeof(int));
                        toSuggestWordIndexes1Distance[oneDistanceExist] = i;
                        oneDistanceExist++;
                }
                //eger matris sonuna ulasinca iki kelime arasindaki mesafe 2 ise
                else if (distanceMatrix[strlen(word)][strlen(dictionary[i])] == 2)
                {
                        //ilgili kelimenin sozlukteki indisini, 2 mesafeli kelime onerilerinin
oldugu diziye ekleriz.
                        toSuggestWordIndexes2Distance =
(int*)realloc(toSuggestWordIndexes2Distance, (twoDistanceExist + 1) * sizeof(int));
                        toSuggestWordIndexes2Distance[twoDistanceExist] = i;
                        twoDistanceExist++;
                }
        }
    }
    //eger 1 mesafeli kelime bulduysak
    if (oneDistanceExist)
    {
        //ilgili kelimeleri kullaniciya oneririz
        printf(" Did you mean: ");
        for (i = 0; i < oneDistanceExist; i++)
        {
            printf("'%s' ", dictionary[toSuggestWordIndexes1Distance[i]]);
        }

        printf("\nEnter the chosen word: ");
        //kullanicidan girdi aliriz.
        scanf("%s", input_word);
        for (i = 0; i < oneDistanceExist; i++)
        {
            //kullanicinin girdigi kelime onerdiklerimiz arasinda var mi?
            if (strcmp(dictionary[toSuggestWordIndexes1Distance[i]], input_word) == 0)
            {
                //var ise sozlukteki indisini dondururuz.
                return toSuggestWordIndexes1Distance[i];
            }
        }
        //yok ise, kullanicinin kendi yazdigi kelimeyi kabul ettigimizi belirtmek icin -1
doneriz.
        return -1;
    }
    //eger 1 mesafeli kelime bulamadiysak ve 2 mesafeli kelime bulduysak
    else if (twoDistanceExist)
    {
        //ilgili kelimeleri kullaniciya oneririz
        printf(" Did you mean: ");
        for (i = 0; i < twoDistanceExist; i++)
        {
            printf("'%s' ", dictionary[toSuggestWordIndexes2Distance[i]]);
        }

        printf("\nEnter the chosen word: ");
        //kullanicidan girdi aliriz.
        scanf("%s", input_word);
        for (i = 0; i < twoDistanceExist; i++)
        {
            //kullanicinin girdigi kelime onerdiklerimiz arasinda var mi?
```

```
                        if (strcmp(dictionary[toSuggestWordIndexes2Distance[i]], input_word) == 0)
                        {
                                //var ise sozlukteki indisini dondururuz.
                                return toSuggestWordIndexes2Distance[i];
                        }
                }
                //yok ise, kullanicinin kendi yazdigi kelimeyi kabul ettigimizi belirtmek icin -1
doneriz.
                return -1;
        }
        //eger 1 veya 2 mesafeli kelime yoksa,
        else
        {
                //sozlukte o kelimeyi bulamadigimizi bildirmek amaciyla -2 doneriz.
                printf(" Couldn't find similar word in dictionary too.\n");
                return -2;
        }
}

//eger kullanici, kendisine onerilen kelimeler arasindan birini secerse, yanlis girilen kelime ve
//kullanicinin oneriler arasindan sectigi kelimeyi yanlis kelimelerin oldugu sozluge ekleyen
fonksiyon
void addToWrongWordsDictionary(WORD_COUPLE* wrong_words_dictionary, char* word, char*
corrected_word)
{
        int result_address;//yanlis kelimeler sozlugunde aradigimiz kelime varsa indisi bu degiskende
saklariz
        result_address = searchWord2(wrong_words_dictionary, word);
        //eger searchWord2 fonksiyonu 0'dan kucuk bir deger donmusse, o degerin pozitifi, bos olan
indistir. fonksiyonumu buna gore kurguladim.
        if (result_address < 0)
        {
                result_address *= -1;

                wrong_words_dictionary[result_address].wrong_word = (char*)calloc(strlen(word),
sizeof(char));
                strcpy(wrong_words_dictionary[result_address].wrong_word, word);

                wrong_words_dictionary[result_address].correct_word =
(char*)calloc(strlen(corrected_word), sizeof(char));
                strcpy(wrong_words_dictionary[result_address].correct_word, corrected_word);
        }
        else
        {
                //eger 0'dan kucuk deger donmemisse M + 1 donmustur. bu da demektir ki yanlis
kelimeler sozlugu dolu.
                printf("Can't add '%s' to wrong words dictionary.. Wrong words dictionary is
full..\n", word);
        }
}

//kullanicidan yeni bir cumle alip bu cumlenin her bir kelimesi icin;
//1)kelime sozluk tablosunda var mi diye kontrol eder,
//1)A)varsa kelime dogrudur
//1)B)yanlis yazilmissa dogrusunu yanlis kelimeler sozlugunde arar,
//2)A)yanlis kelimeler sozlugunde bulamazsa dogru kelimelerin oldugu sozlukteki kelimelerle
arasindaki mesafeyi bulmak icin gerekli fonksiyonlari cagirir.
//2)B)yanlis kelimeler sozlugunde varsa, ona karsilik gelen dogru kelimeyi kullaniciya onerir.
//en son olarak da cumlenin duzeltilmis halini ekrana yazdirir.
void getNewSentence(char** dictionary, WORD_COUPLE* wrong_words_dictionary)
{
        char sentence[MAX_SENTENCE_LENGTH];//kullanicidan alinan cumleyi saklamak icin degisken
        char* sentence_corrected;//cumlenin yeni ve duzeltilmis halini saklamak icin degisken
        char* word;//o anki kelime degiskeni.
        char input_word[MAX_WORD_LENGTH];//kullaniciya onerilen kelimeler arasindan bir secim
yaptirmak icin bu degisken kullanilir.
```

```
	int result_address;//cagrilan fonksiyonlarin donus degerleri tutulur
	printf("Enter the sentence without special characters: ");
	//bi bug sebebiyle kullandim
	scanf("%c");
	//kullanicidan cumle alinir.
	fgets(sentence, MAX_SENTENCE_LENGTH, stdin);
	//alinan cumlenin terminating karakteri elle ayarlanir. bu islem yapilmazsa \n karakteri de
cumleye dahil ediliyordu.
	if ((strlen(sentence) > 0) && (sentence[strlen(sentence) - 1] == '\n'))
	{
		sentence[strlen(sentence) - 1] = '\0';
	}
	//strtok fonksiyonu ile cumlenin ilk kelimesi alinir.
	word = strtok(sentence, " ");
	//baslangic olarak 1 harflik yer acilir.
	sentence_corrected = (char*)calloc(1, sizeof(char));
	//cumlenin sonuna kadar bu dongude kalinir.
	while (word != NULL)
	{
		//kelime sozlukte var mi diye kontrol et.
		if (searchWord(dictionary, word) != 0)
		{
			//kelime sozlukte yok, hatali kelime tablosunda ara.
			result_address = searchWord2(wrong_words_dictionary, word);
			//donus degeri 0'dan kucukse veya M'den buyukse hatali kelime tablosunda da
yoktur.
			if ((result_address < 0) || (result_address > M))
			{
				//hatali kelime tablosunda yok, sozluk tablosundaki en yakin kelimeyi
bul.
				printf("'%s' is not in the dictionary.", word);
				result_address = editDistance(dictionary, word);

				//eger -2 donerse, girilen kelimeye yakin bir kelime bulunamamistir.
kullanicinin girdigi kelime kabul edilip ekrana yazilir.
				if (result_address == -2)
				{
					sentence_corrected = (char*)realloc(sentence_corrected,
(strlen(word) + strlen(sentence_corrected) + 2) * sizeof(char));
					strcat(sentence_corrected, word);
					strcat(sentence_corrected, " ");
				}
				//eger -1 donerse, kullanici onerilen kelimeleri kabul etmeyip kendi
yazdigi kelimeyi kabul etmistir. kullanicinin yazdigi kelime ekrana yazilir.
				else if (result_address == -1)
				{
					sentence_corrected = (char*)realloc(sentence_corrected,
(strlen(word) + strlen(sentence_corrected) + 2) * sizeof(char));
					strcat(sentence_corrected, word);
					strcat(sentence_corrected, " ");
				}
				//eger -2 veya -1'den farkli bir deger donerse, kullanici onerilen
kelimeler arasindan birini secmistir. bu kelime ekrana yazilir ve hatali kelimeler
				//tablosuna eklenir.
				else
				{
					sentence_corrected = (char*)realloc(sentence_corrected,
(strlen(dictionary[result_address]) + strlen(sentence_corrected) + 2) * sizeof(char));
					strcat(sentence_corrected, dictionary[result_address]);
					strcat(sentence_corrected, " ");
					addToWrongWordsDictionary(wrong_words_dictionary, word,
dictionary[result_address]);
				}
			}
			else
			{
```

```
                        //hatali kelime tablosunda var, oradaki kabul gormus kelimeyi oner.
                        printf("'%s' is not in the dictionary. Did you mean: '%s'\nEnter the
chosen word: ", word, wrong_words_dictionary[result_address].correct_word);
                        scanf("%s", input_word);
                        if (strcmp(input_word,
wrong_words_dictionary[result_address].correct_word) == 0)
                        {
                                //kullanici onerilen kelimeyi kabul etti, duzeltilmis cumle
degiskenine kopyala.
                                sentence_corrected = (char*)realloc(sentence_corrected,
(strlen(wrong_words_dictionary[result_address].correct_word) + strlen(sentence_corrected) + 2) *
sizeof(char));
                                strcat(sentence_corrected,
wrong_words_dictionary[result_address].correct_word);
                                strcat(sentence_corrected, " ");
                        }
                        else
                        {
                                //kullanici onerilen kelimeyi kabul etmedi, kullanicinin girdigi
'hatali' kelimeyi ekrana yazdir.
                                sentence_corrected = (char*)realloc(sentence_corrected,
(strlen(word) + strlen(sentence_corrected) + 2) * sizeof(char));
                                strcat(sentence_corrected, word);
                                strcat(sentence_corrected, " ");
                        }
                }
        }
        else
        {
                //kelime sozlukte var, duzeltilmis cumle degiskenine kopyalariz.
                sentence_corrected = (char*)realloc(sentence_corrected, (strlen(word) +
strlen(sentence_corrected) + 2) * sizeof(char));
                strcat(sentence_corrected, word);
                strcat(sentence_corrected, " ");
        }
        word = strtok(NULL, " ");
    }
    //duzeltilmis cumle degiskeninin en sonundaki fazladan bosluk karakterini silip yerine
terminating character koyariz.
    sentence_corrected[strlen(sentence_corrected)-1] = '\0';
    printf("\nNew version of given sentence: '%s'\n\n", sentence_corrected);
}

//sozluk dosyasindaki kelimelerle dictionary tablosunu dolduran fonksiyon
void populateDictionary(char** dictionary)
{
    FILE* fp;//dosya degiskeni
    char ch;//dosya sonuna ulasip ulasmadigimizi kontrol etmek icin
    int i = 0;//dongu degiskeni
    int key;//key degiskeni
    char word[MAX_WORD_LENGTH];//dosyadan alinan siradaki kelimeyi saklamak icin degisken
    int address;//tek bir adimda hash ile bos hucre bulabilirsek bu degiskene indisi atariz
    int finalAddress;//bir adimda bulamazsak double hash yontemiyle buldugumuz bos hucrenin
indisini bu degiskende saklariz

    //dosya acma islemi.
    fp = fopen(DICTIONARY_FILE_NAME, "r");
    if (!fp)
    {
        printf("Couldn't open dictionary file..Exiting..\n");
        exit(0);
    }

    //sozluk boyutu kadar bu dongude kaliriz
    for (i = 0; i < M; i++)
    {
```

```
		//kelime alinir
		ch = fscanf(fp, "%s", word);
		//ch EOF olmussa dosya sonuna gelmisizdir. fonksiyondan cikariz.
		if (ch == EOF)
		{
			return;
		}
		//key ve address uretimi.
		key = produceKey(word);
		address = hash1(key);


		//sozlugun ilgili hucresi bos mu?
		if (dictionary[address] != NULL)
		{
			//hucre bos degil, kelimeyle hucredeki kelime ayni mi?
			if (strcmp(dictionary[address], word) != 0)
			{
				//kelimeler farkli, bos bir hucre veya o kelimenin oldugu bir hucre
bulana kadar while dongusunde kaliriz.
				i = 1;
				while (1)
				{
					finalAddress = doubleHash(key, i);

					//bos bir hucre bulduysak, kelime eklenir.
					if (dictionary[finalAddress] == NULL)
					{
						dictionary[finalAddress] = (char*)calloc(strlen(word),
sizeof(char));
						strcpy(dictionary[finalAddress], word);
						break;
					}
					//eklemek itedigimiz kelimeyi sozlukte bulduysak tekrar eklemeyiz.
					else if (strcmp(dictionary[finalAddress], word) == 0)
					{
						break;
					}
					i++;
				}
			}
		}
		else
		{
			//tablonun o hucresi bos. kelime eklenir.
			dictionary[address] = (char*)calloc(strlen(word), sizeof(char));
			strcpy(dictionary[address], word);
		}
	}
}

int main()
{
	char** dictionary;//sozluk tablosu
	WORD_COUPLE* wrong_words_dictionary; //hatali kelimelerin ve onlara karsilik gelen kabul
edilmis dogru kelimelerin oldugu sozluk tablosu
	int opCode;//kullanicidan alinan islem kodu, 0 programdan cikar, 1 yeni cumle alir.

	//iki tabloya da M kadar yer acilir.
	dictionary = (char**)calloc(M, sizeof(char*));
	wrong_words_dictionary = (WORD_COUPLE*)calloc(M, sizeof(WORD_COUPLE));

	populateDictionary(dictionary);

	do
	{
```

```
            printf("0-Exit\n1-Enter new sentence\n");
            printf("Enter operation code: ");
            scanf("%d", &opCode);
            switch (opCode)
            {
            case 0:
                  return 0;
            case 1:
                  getNewSentence(dictionary, wrong_words_dictionary);
                  break;
            }
      } while (opCode);
            return 0;
}
```