YILDIZ TEKNİK ÜNİVERSİTESİ
ELEKTRİK-ELEKTRONİK FAKÜLTESİ
BİLGİSAYAR MÜHENDİSLİĞİ BÖLÜMÜ

2020-2021 GÜZ YARIYILI

BLM3021 ALGORİTMA ANALİZİ DERSİ

GRUP-2

2.ÖDEV RAPORU

Konu: Hashing Algoritmasının Kullanımı

Hazırlayan: Mehmet Hayri Çakır – 16011023

Dersin Yürütücüsü: Dr. Öğr. Üyesi Mehmet Amaç GÜVENSAN

İstanbul, Aralık 2020

İçindekiler

1. Yöntem

1.1. Problem

Bu ödevde, bir kelimenin geçtiği dokümanları listeleyen bir sistem tasarlanacaktır. Bir kelimenin hangi dokümanlarda geçtiğini bulmak için her seferinde bütün dokümanlara bakmak çok zaman alıcıdır.

1.2. Çözüm

Bunun yerine bu ödevde, yeni gelen bir dokümandaki kelimeler hashing ile bir sözlüğe yerleştirilecek ve bir kelime arandığında yine hashing yöntemi ile sözlükte aranarak içinde yer aldığı dokümanlar bulunacaktır. Hash tablosunu oluştururken open addressing, çakışma problemini çözmek için double hashing yöntemleri kullanılmıştır. Key üretilirken Horner metodu kullanılmıştır.

2. Uygulama

2.1. Fonksiyonlar

Main fonksiyonu hariç toplam 14 fonksiyon kullanılmıştır:

- `int hash1(int key)`
- `int hash2(int key)`
- `int doubleHash(int key, int i)`
- `int produceKey(char* word)`
- `void createHashFile()`
- `FILE* openHashFile(char *openMode)`
- `void updateHashTable(WORD_INFO* hashTable, char *fileName, int *currentSize, float *loadFactor)`
- `void updateHash(WORD_INFO* hashTable, int address, char *fileName, int *currentSize, float *loadFactor)`
- `void insertHash(WORD_INFO* hashTable, int address, int key, char word[32], char *fileName, int *currentSize, float *loadFactor)`
- `void readHashFile(FILE* hashFile, WORD_INFO* hashTable, int *currentSize, float *loadFactor)`
- `void writeToHashFile(WORD_INFO *hashTable, int *currentSize, float *loadFactor)`
- `int searchWord(WORD_INFO* hashTable, char* word)`
- `void printTable(WORD_INFO* hashTable)`
- `void maxStepsToFindAWord(WORD_INFO* hashTable)`

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. void createHashFile()

Eğer hash tablosunun yazılacağı dosya mevcut değilse bir kereye mahsus bu fonksiyon çalışır ve dosya oluşturulur. Dosya silinmediği veya program başka bir konuma taşınmadığı sürece sonraki çalıştırmalarda bu fonksiyon çağrılmaz.

2.1.6. FILE* openHashFile(char *openMode)

Eğer hash tablosunun yazıldığı dosya diskte mevcutsa bu fonksiyon ile açılır.

2.1.7. void readHashFile(FILE* hashFile, WORD_INFO* hashTable, int *currentSize, float *loadFactor)

Program çalıştırıldığında *openHashFile* fonksiyonu ile açılan dosyadaki bilgiler hash tablosuna bu fonksiyon ile yazılır.

2.1.8. void writeToHashFile(WORD_INFO *hashTable, int *currentSize, float *loadFactor)

Program, menüdeki çıkış fonksiyonu ile kapatılırsa (varsayılan tuş 0'dır) hash tablosundaki bilgileri diskteki dosyaya yazar. Dosya adı kodun başında define ile belirtilmiştir.

2.1.9. void updateHashTable(WORD_INFO* hashTable, char *fileName, int *currentSize, float *loadFactor)

Programa, yeni doküman verilmesi durumunda bu fonksiyon çağrılır. Tabloda kelimenin olup olmamasına göre *insertHash* veya *updateHash* fonksiyonu çağrılır.

2.1.10. void insertHash(WORD_INFO* hashTable, int address, int key, char word[32], char *fileName, int *currentSize, float *loadFactor)

Kelime tabloda yoksa bu fonksiyon çağrılır. Kelime ve geçtiği doküman tabloya eklenir.

2.1.11. void updateHash(WORD_INFO* hashTable, int address, char *fileName, int *currentSize, float *loadFactor)

Kelime tabloda mevcutsa sadece geçtiği dokümanlar kısmı güncellenerek şuanda okunan dokümanın adı eklenir.

2.1.12. int searchWord(WORD_INFO* hashTable, char* word)

Tabloda bir kelime aranmak istenirse bu fonksiyon çağrılır. Kelime tabloda bulunursa, kaç adımda bulunduğu ve geçtiği doküman isimleri ekrana yazdırılır.

2.1.13. void printTable(WORD_INFO* hashTable)

Hash tablosunun her indisinin içeriğini ekrana yazdırmak istendiğinde bu fonksiyon kullanılabilir. Hangi kelime o adreste saklanıyor, bu kelime kaç dokümanda geçiyor ve bu dokümanların isimlerini ekrana yazdırır.

2.1.14. void maxStepsToFindAWord(WORD_INFO* hashTable)

Tabloda yer alan kelimelerin her biri için; hangi indiste yer aldığı, kaç adımda bulunduğu, kelimenin kendisi ve bulunduğu dokümanları ekrana yazdırır. En son olarak tabloda mevcut olan bir kelimeyi bulmanın en fazla kaç adımda tamamlandığını ekrana yazdırır.

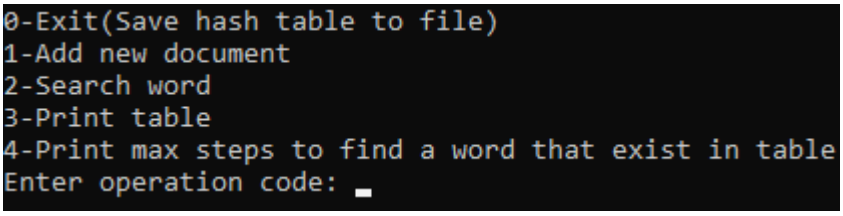
2.1.15. int main()

Program ilk çalıştığında hash tablosuna yer açmak, varsa diskteki hash dosyasını açmak yoksa oluşturmak işlemleri gerçekleştirilir. Sonrasında switch-case yapısı ile kullanıcıdan alınan girdilere karşılık gelen fonksiyonlar çağrılır. Yapılabilecek işlemler; hash tablosunu diske kaydedip programı kapatmak, yeni doküman eklemek, tabloda kelime aramak, tabloyu yazdırmak, tabloda bulunan bir kelimeyi bulmak için gereken maksimum adım sayısı…

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. updateHashTable (Yeni doküman eklenmesi)

2.2.2.1. Tabloya yeni kelime eklenmesi

Sırasıyla; tablonun dolu hücre sayısı (tablonun mevcut boyutu, currentSize değişkeni), eklenen kelime, load factor ekrana yazdırılır.

```
0-Exit(Save hash table to file)
1-Add new document
2-Search word
3-Print table
4-Print max steps to find a word that exist in table
Enter operation code: 1
Enter document name in the format of (example.txt): 2.txt
  1|Word cakir                was not present in hashtable, added... Load factor: 0.001003
  2|Word hayri                was not present in hashtable, added... Load factor: 0.002006
```

2.2.2.2. Tabloda olan kelimelere yeni dokümanın adının eklenmesi

Tabloya yeni kelime eklenmediği için dolu hücre sayısı(2) güncellenmedi.

```
0-Exit(Save hash table to file)
1-Add new document
2-Search word
3-Print table
4-Print max steps to find a word that exist in table
Enter operation code: 1
Enter document name in the format of (example.txt): 3.txt
  2|Word cakir                is present in hashtable, added new document name... Load factor: 0.002006
  2|Word hayri                is present in hashtable, added new document name... Load factor: 0.002006
```

2.2.2.3. Tablo doluyken yeni kelime eklenmesi

Tabloya eklenen “olan” kelimesiyle birlikte tablo dolar ve load factor 1 olur. Sonrasında gelen “ama” kelimesi tabloda aranır ve bulunamadığı için ekrana “Word does not exist in hashtable…” şeklinde bilgi yazısı basılır. Eğer “ama” kelimesi tabloda olsaydı, sadece yeni doküman adı ekleneceği için problem olmayacaktı.

```
997|Word olan                was not present in hashtable, added... Load factor: 1.000000
Load factor is above 0.800000 limit...
Search completed in 997 steps...Word:   ama             does not exist in hashtable...
Can't insert the word 'ama'. Load factor is: 1.000000
```

2.2.2.4. Tablo doluyken tabloda mevcut bir kelime eklenmesi

Tabloya eklenen “olan” kelimesiyle birlikte tablo dolar ve load factor 1 olur. Sonrasında gelen “ama” kelimesi tabloda aranır ve bulunduğu için ekrana bilgi yazısı basılır. Sadece yeni doküman adı ekleneceği için problem olmaz. Tabloya yeni kelime eklenmediği için dolu hücre sayısı(997) güncellenmedi.

```
997|Word olan                was not present in hashtable, added... Load factor: 1.000000
Load factor is above 0.800000 limit...
Search completed in 121 steps...Word:   ama             found in the following files: master.txt
997|Word ama                is present in hashtable, added new document name... Load factor: 1.000000
```

2.2.3. searchWord (Tabloda kelime aranması)

2.2.3.1. Bir dokümanda geçen kelimenin aranması

```
0-Exit(Save hash table to file)
1-Add new document
2-Search word
3-Print table
4-Print max steps to find a word that exist in table
Enter operation code: 2
Enter word to search in table: mehmet
Search completed in   1 steps...Word:   mehmet          found in the following files: 3.txt
```

2.2.3.2. Birden fazla dokümanda geçen kelimenin aranması

```
0-Exit(Save hash table to file)
1-Add new document
2-Search word
3-Print table
4-Print max steps to find a word that exist in table
Enter operation code: 2
Enter word to search in table: hayri
Search completed in   1 steps...Word:   hayri          found in the following files: 2.txt 3.txt
```

2.2.3.3. Tabloda bulunmayan kelimenin aranması

```
0-Exit(Save hash table to file)
1-Add new document
2-Search word
3-Print table
4-Print max steps to find a word that exist in table
Enter operation code: 2
Enter word to search in table: selam
Search completed in   1 steps...Word:   selam          does not exist in hashtable...
```

2.2.3.4. Tablo doluyken tabloda bulunmayan kelimenin aranması

```
0-Exit(Save hash table to file)
1-Add new document
2-Search word
3-Print table
4-Print max steps to find a word that exist in table
Enter operation code: 2
Enter word to search in table: selam
Search completed in 997 steps...Word:   selam          does not exist in hashtable...
```

2.2.4. printTable (Tablonun yazdırılması)

Sırasıyla; tablo indisi, kelimenin kaç tane dokümanda geçtiği, kelimenin kendisi ve geçtiği dokümanların isimleri yazdırılır.

```
279|0  @
280|0  @
281|2  hayri                2.txt 3.txt
```

2.2.5. maxStepsToFindAWord

Sırasıyla; tablo indisi, o indiste kelime mevcutsa kaç adımda bulunduğu, kelimenin kendisi, bulunduğu dokümanlar yazdırılır. Tablonun o indisi boşsa, boş olduğuna dair kullanıcıya bilgi verilir.

```
976|Slot is empty...
977|Search completed in   2 steps...Word:       bright         found in the following files: master.txt
978|Search completed in   1 steps...Word:       warning        found in the following files: master.txt
979|Search completed in   3 steps...Word:       important      found in the following files: master.txt
980|Search completed in   1 steps...Word:       inserted       found in the following files: master.txt
981|Search completed in   3 steps...Word:       bed.           found in the following files: master.txt
982|Search completed in   1 steps...Word:       instead,       found in the following files: master.txt
983|Search completed in   1 steps...Word:       discover       found in the following files: master.txt
984|Search completed in   1 steps...Word:       spelling       found in the following files: master.txt
985|Search completed in   1 steps...Word:       product        found in the following files: master.txt
986|Search completed in   1 steps...Word:       program        found in the following files: master.txt
987|Search completed in   2 steps...Word:       where          found in the following files: master.txt
988|Search completed in   1 steps...Word:       darkness,      found in the following files: master.txt
989|Search completed in   2 steps...Word:       without,       found in the following files: master.txt
990|Search completed in   3 steps...Word:       though         found in the following files: master.txt
991|Search completed in   3 steps...Word:       bring          found in the following files: master.txt
992|Search completed in   1 steps...Word:       concept        found in the following files: master.txt
993|Search completed in   1 steps...Word:       you're,        found in the following files: master.txt
994|Search completed in   1 steps...Word:       transposed     found in the following files: master.txt
995|Search completed in   4 steps...Word:       life.          found in the following files: master.txt
996|Search completed in  22 steps...Word:       mountains,     found in the following files: master.txt
Max steps to find a word: 122
Word:   ama
```

3. Kod

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

#define R 2 //horner metodundaki R degiskeni
#define HASH_FILE_NAME "hashFile.txt" //hash tablosunun bulundugu/yazilacagi dosyanin adi
#define HASH_TABLE_EMPTY_SLOT_CHAR "@" //tablonun bos hucrelerini anlamak icin kullanilan karakter
#define MAX_FILE_NAME_LENGTH 32 //maksimum dosya adi uzunlugu
#define MAX_WORD_LENGTH 32 // maksimum kelime uzunlugu
#define M 997 //tablo boyutu, 1000'e en yakin asal sayi istenmisti
#define LOAD_FACTOR_WARNING_LIMIT 0.8 //load factor bu degeri gectiginde ekrana uyari yazilacak.

typedef struct
{
	//gerek olmadigi icin key degiskenini sildim. islemlerimi word degiskeni uzerinden yaptim.
	//int key;
	char* word; //kelime
	char** documents; //gectigi dokuman isimleri
	short documentsCount; //kac dokumanda gectigi
}WORD_INFO; //tablonun tipi.


int hash1(int key);
int hash2(int key);
int doubleHash(int key, int i);
int produceKey(char* word);
void createHashFile();
FILE* openHashFile(char *openMode);
void updateHashTable(WORD_INFO* hashTable, char *fileName, int *currentSize, float *loadFactor);
void updateHash(WORD_INFO* hashTable, int address, char *fileName, int *currentSize, float
*loadFactor);
void insertHash(WORD_INFO* hashTable, int address, int key, char word[32], char *fileName, int
*currentSize, float *loadFactor);
void readHashFile(FILE* hashFile, WORD_INFO* hashTable, int *currentSize, float *loadFactor);
void writeToHashFile(WORD_INFO *hashTable, int *currentSize, float *loadFactor);
int searchWord(WORD_INFO* hashTable, char* word);
void printTable(WORD_INFO* hashTable);
void maxStepsToFindAWord(WORD_INFO* hashTable);

int main()
{
	WORD_INFO* hashTable; //hash tablosu
	FILE* hashFile; //hash tablosunun bulundugu/yazilacagi dosya
	char fileName[MAX_FILE_NAME_LENGTH];//yeni dokuman eklenecegi zaman dosya adi bu degiskene
scanf ile alinir
	char word[MAX_WORD_LENGTH];//kelime aranacagi zaman bu degiskene scanf ile alinir
	int opCode;//yapilmak istenen islemin kodu, 0 girilene kadar program loop'ta calisir. 0
girildiginde hash tablosu dosyaya yazilip program kapatilir.
	int *currentSize;//tablonun dolu olan hucrelerinin sayisi
	int i;//dongu degiskeni
	float *loadFactor;//tablo doluluk orani

	hashTable = (WORD_INFO*)calloc(M, sizeof(WORD_INFO)); //hash tablosuna yer acilmasi
	for (i = 0; i < M; i++)
	{
		//hashTable[i].key = -1;
		//tablonun tum hucrelerine bos oldugunu bildiren karakterin yerlestirilmesi
		hashTable[i].word = (char*)malloc(strlen(HASH_TABLE_EMPTY_SLOT_CHAR) * sizeof(char));
		hashTable[i].documentsCount = 0;
		strcpy(hashTable[i].word, HASH_TABLE_EMPTY_SLOT_CHAR);
	}
	currentSize = (int*)calloc(1, sizeof(int));
```

```
	loadFactor = (float*)calloc(1, sizeof(float));

	//hash dosyasini acmaya calisiriz.
	if ((hashFile = openHashFile("r")) == NULL)
	{
		//dosya acma basarisiz olduysa hash dosyasi yoktur. olustururuz ve tabloyu dosyaya yazariz.
		createHashFile();
		writeToHashFile(hashTable, currentSize, loadFactor);
	}
	else
	{
		//dosya acma basarili oldu. demek ki dosya var. kapatiriz.
		fclose(hashFile);
		//dosyadaki verileri tabloya gecirirz.
		readHashFile(hashFile, hashTable, currentSize, loadFactor);
	}

	//program ana dongusu.
	do
	{
		printf("0-Exit(Save hash table to file)\n1-Add new document\n2-Search word\n3-Print table\n4-Print max steps to find a word that exist in table\n");
		printf("Enter operation code: ");
		scanf("%d", &opCode);
		switch (opCode)
		{
		case 0:
			//0 girildiyse while kosulu artik saglanmaz. tabloyu dosyaya yazip programi kapatiriz.
			writeToHashFile(hashTable, currentSize, loadFactor);
			exit(0);
			break;
		case 1:
			//yeni dosya eklenmek isteniyor. dosya adini alip ilgili fonksiyonu cagiririz.
			printf("Enter document name in the format of (example.txt): ");
			scanf("%s", &fileName);
			updateHashTable(hashTable, fileName, currentSize, loadFactor);
			break;
		case 2:
			//tabloda kelime aranmak isteniyor. kelimeyi alip ilgili fonksiyonu cagiririz.
			printf("Enter word to search in table: ");
			scanf("%s", &word);
			searchWord(hashTable, word);
			break;
		case 3:
			//tabloyu yazdirmak istendi. ilgili fonksiyonu cagiririz.
			printTable(hashTable);
			break;
		case 4:
			//tabloda mevcut olan bir kelimeyi aramak icin gecen maksimum adim sayisini buluruz.
			maxStepsToFindAWord(hashTable);
			break;
		default:
			break;
		}
		printf("\n\n");
	} while (1);

	free(hashTable);
	return 0;
}

//tabloda mevcut olan bir kelimeyi bulmak icin gecen maksimum adim sayisini bulur
void maxStepsToFindAWord(WORD_INFO *hashTable)
```

```
{
	int i; //dongu degiskeni
	int maxSteps = 0; //suanki maksimum adim, bundan fazla adim suren kelime bulunmasi durumunda guncellenir.
	int currentSteps; //suanki kelimeyi bulmanin kac adim surdugu bu degiskende saklanir
	char word[MAX_WORD_LENGTH]; //en fazla adimda bulunan kelime
	for (i = 0; i < M; i++)
	{
		if (strcmp(hashTable[i].word, HASH_TABLE_EMPTY_SLOT_CHAR))
		{
			printf("%3d|", i);
			if ((currentSteps = searchWord(hashTable, hashTable[i].word)) > maxSteps)
			{
				maxSteps = currentSteps;
				strcpy(word, hashTable[i].word);
			}
		}
		else
		{
			printf("%3d|Slot is empty...\n", i);
		}
	}

	printf("Max steps to find a word: %3d \nWord:\t%-15s\n", maxSteps, word);
}

//tablonun her hucresinin icerigini yazdirir.
void printTable(WORD_INFO* hashTable)
{
	int i, j;
	for (i = 0; i < M; i++)
	{
		printf("%-3d|%-3d%-20s ", i, hashTable[i].documentsCount, hashTable[i].word);
		for (j = 0; j < hashTable[i].documentsCount; j++)
		{
			printf("%s ", hashTable[i].documents[j]);
		}
		printf("\n");
	}
}

//tabloda bir kelime aramak icin kullanilir.
int searchWord(WORD_INFO* hashTable, 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); //tablonun indisini belirten degisken

	//odev dosyasında program buyuk kucuk harf duyarsiz dendigi icin kelime kucuk harfe donusturulur.
	for (i = 0; i < strlen(word); i++)
	{
		word[i] = tolower(word[i]);
	}

	//tablonun o hucresi bos mu?
	if (strcmp(hashTable[address].word, HASH_TABLE_EMPTY_SLOT_CHAR) == 0)
	{
		//bossa kelime tabloda yoktur
		printf("Search completed in %3d steps...Word:\t%-15s does not exist in hashtable...\n", 1, word);
		return -1;
	}
	//bos degilse aradigimiz kelime o hucredeki kelime mi?
	else if(strcmp(hashTable[address].word, word) == 0)
```

```
	{
		//cevap evetse kelimeyi bulduk. ilgili yazdirma islemleri yapilir. kac adimda buldugumuz(1) dondurulur.
		printf("Search completed in %3d steps...", 1);
		printf("Word:\t%-15s found in the following files: ", word);

		for (i = 0; i < hashTable[address].documentsCount; i++)
		{
			printf("%s ", hashTable[address].documents[i]);
		}
		printf("\n");
		return 1;
	}
	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 (strcmp(hashTable[address].word, word) && (strcmp(hashTable[address].word, HASH_TABLE_EMPTY_SLOT_CHAR)))
		{
			address = doubleHash(key, i);
			i++;
			if (i == M)
			{
				//i tablo boyutunu asarsa kelime tabloda yoktur. (tablonun dolu oldugu durum icin sonsuz donguden kacinma amacli yazdim)
				printf("Search completed in %3d steps...Word:\t%-15s does not exist in hashtable...\n", i, word);
				return -1;
			}
		}
		//hucredeki kelime aradigimiz kelime mi?
		if (strcmp(hashTable[address].word, word) == 0)
		{
			//cevap evetse aradigimiz kelimeyi bulduk, ilgili yazdirma islemleri yapilip kac adimda buldugumuzu doneriz.
			printf("Search completed in %3d steps...", i);
			printf("Word:\t%-15s found in the following files: ", word);

			for (j = 0; j < hashTable[address].documentsCount; j++)
			{
				printf("%s ", hashTable[address].documents[j]);
			}
			printf("\n");
			return i + 1;
		}
		else
		{
			//bos hucreye denk geldik. kelime tabloda yoktur.
			printf("Search completed in %3d steps...Word:\t%-15s does not exist in hashtable...\n", i, word);
			return -1;
		}
	}
}

//yeni dokuman geldiginde bu fonksiyon cagrilir
void updateHashTable(WORD_INFO* hashTable, char* fileName, int *currentSize, float *loadFactor)
{
	FILE* document; //eklenmek istenen dokuman pointeri
	int key; //dokumandaki kelimelerin key karsiliklari icin kullanilir.
	int finalAddress; //tabloya erismek icin indis
	int address; //tabloya erismek icin indis
	int i; //dongu degiskeni
```

```
	int j; //dongu degiskeni
	int k; //dongu degiskeni
	char ch; //dokuman okunurken EOF kontrolu icin
	char word[MAX_WORD_LENGTH]; //dokumandan okunan kelime icin

	//dokuman mevcut degilse fonksiyondan donulur.
	if (!(document = fopen(fileName, "r")))
	{
		printf("Can't open file '%s'", fileName);
		return;
	}

	//ilk kelime alinir
	ch = fscanf(document, "%s", word);

	//dosya sonuna gelene kadar dongude kaliriz.
	while (ch != EOF)
	{
		//tablo dolduysa ve eklenmek istenen kelime tabloda mevcut degilse daha fazla kelime ekleyemeyiz. kelime tabloda mevcutsa o adresteki struct'a yeni dokuman adini ekleriz sadece.
		if ((*currentSize) == M && (searchWord(hashTable, word) == -1))
		{
			printf("Can't insert the word '%s'. Load factor is: %f\n", word, *loadFactor);
			ch = fscanf(document, "%s", word);
			continue;
		}

		key = produceKey(word);
		address = hash1(key);

		//tablonun ilgili hucresi bos mu?
		if (strcmp(hashTable[address].word, HASH_TABLE_EMPTY_SLOT_CHAR))
		{
			//hucre bos degil, eklenmek istenen kelimeyle hucredeki kelime ayni mi?
			if (strcmp(hashTable[address].word, word) == 0)
			{
				//eger ayniysa hucre guncellenerek yeni dokuman adi eklenir.
				updateHash(hashTable, address, fileName, currentSize, loadFactor);
			}
			else
			{
				//eger kelimeler farkliysa 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 insertHash ile kelime ve dokuman adi eklenir.
					if (strcmp(hashTable[finalAddress].word, HASH_TABLE_EMPTY_SLOT_CHAR) == 0)
					{
						insertHash(hashTable, finalAddress, key, word, fileName, currentSize, loadFactor);
						break;
					}
					//eklemek itedigimiz kelimeyi tabloda bulduysak sadece dokuman adini eklemek icin updateHash kullanilir.
					else if (strcmp(hashTable[finalAddress].word, word) == 0)
					{
						updateHash(hashTable, finalAddress, fileName, currentSize, loadFactor);
						break;
					}
					i++;
```

```
                        }
                }
        }
        else
        {
                //tablonun o hucresi bos. kelime ve dokuman adi eklenir.
                insertHash(hashTable, address, key, word, fileName, currentSize, loadFactor);
        }
        ch = fscanf(document, "%s", word);
    }
    fclose(document);
}

//eklenmek istenen kelime tabloda mevcut, sadece dokuman adini ekleriz.
void updateHash(WORD_INFO* hashTable, int address, char* fileName, int* currentSize, float*
loadFactor)
{
    int i = 0; //dongu degiskeni

    //dokuman adi zaten o hucrede mevcut mu diye kontrol ederiz. (bir kelime ayni dokumanda 1'den
fazla yerde geciyorsa tekrar tekrar eklenmesini engellemek icin.)
    while ((i < hashTable[address].documentsCount) && (strcmp(hashTable[address].documents[i],
fileName) != 0))
    {
        i++;
    }
    //eger onceki donguden; i, kelimenin gectigi dokuman sayisina esit olmadan ciktiysak, dokuman
adi zaten eklenmistir. fonksiyondan doneriz.
    if (i != hashTable[address].documentsCount)
    {
        return;
    }

    //dokuman adi kadar yer acilir.
    hashTable[address].documents[hashTable[address].documentsCount]                    =
(char*)calloc(strlen(fileName), sizeof(char));

    //dokuman adini hucreye yazma islemi
    strcpy(hashTable[address].documents[hashTable[address].documentsCount], fileName);
    hashTable[address].documentsCount++;
    printf("%3d|Word %-20s is present in hashtable, added new document name... Load factor: %f\n",
*currentSize, hashTable[address].word, *loadFactor);
}

//eklenmek istenen kelime tabloda yok. kelimeyi ve dokuman adini ekleriz
void insertHash(WORD_INFO* hashTable, int address, int key, char  word[32], char* fileName, int
*currentSize, float *loadFactor)
{
    //hashTable[address].key = key;
    //kelimeye ve dokuman adina yer acma islemleri.
    hashTable[address].word    =    (char*)realloc(hashTable[address].word,    strlen(word)    *
sizeof(char));
    hashTable[address].documents = (char**)malloc(sizeof(char*));
    hashTable[address].documents[hashTable[address].documentsCount]                    =
(char*)calloc(strlen(fileName), sizeof(char));

    //hucreye yazma islemleri.
    strcpy(hashTable[address].word, word);
    strcpy(hashTable[address].documents[hashTable[address].documentsCount], fileName);
    hashTable[address].documentsCount++;

    //tablonun dolu hucre sayisi ve loadfactor guncellenir.
    (*currentSize)++;
    *loadFactor = (float)(*currentSize) / M;

    //eger load faktor define ile tanimlanan degerden buyukse ekrana uyari yazilir.
```

```
		printf("%3d|Word %-20s was not present in hashtable, added... Load factor: %f\n", *currentSize,
word, *loadFactor);
		if (*loadFactor > LOAD_FACTOR_WARNING_LIMIT)
		{
			printf("Load factor is above %f limit...\n", LOAD_FACTOR_WARNING_LIMIT);
		}
}

//diskteki hash dosyasini acma islemi.
FILE* openHashFile(char *openMode)
{
		FILE* fp;
		fp = fopen(HASH_FILE_NAME, openMode);
		return fp;
}

//diskte hash dosyasi yoksa olusturma islemi.
void createHashFile()
{
		FILE* fp;
		if ((fp = fopen(HASH_FILE_NAME, "w")) == NULL)
		{
			printf("Couldn't create hash file...");
			return;
		}
		fclose(fp);
}

//program calistiginda diskteki hash dosyasini hashTable degiskenine yazmak icin kullanilir.
void readHashFile(FILE *hashFile, WORD_INFO *hashTable, int *currentSize, float *loadFactor)
{
		char ch; //dosya sonuna geldik mi diye EOF ile karsilastirilir.
		char str[MAX_FILE_NAME_LENGTH]; //dosyadan okunan input. kelime ve gectigi dokuman adlari
okuurken kullanilir.
		int i; //dongu degiskeni
		int j; //dongu degiskeni
		hashFile = openHashFile("r");

		//hash dosyasi acilamadiysa fonksiyondan cikilir.
		if (hashFile == NULL)
		{
			printf("Couldn't open hash file...");
			return;
		}
		//tablonun dolu hucre sayisi okunur
		ch = fscanf(hashFile, "%d", currentSize);
		//eger dosya bossa fonksiyondan donulur.
		if (ch == EOF)
		{
			return;
		}
		//loadfactor okunur
		fscanf(hashFile, "%f", loadFactor);
		for (i = 0; i < M; i++)
		{
			//fscanf(hashFile, "%d", &(hashTable[i].key));

			//kelime okunur ve ilgili hucredeki word degiskenine yazilir
			fscanf(hashFile, "%s", str);
			hashTable[i].word = (char*)realloc(hashTable[i].word, strlen(str) * sizeof(char));
			strcpy(hashTable[i].word, str);

			//dokuman sayisi okunur
			fscanf(hashFile, "%d", &(hashTable[i].documentsCount));
			//dokuman isimleri icin gerekli yer acilir.
```

```
			hashTable[i].documents = (char**)realloc(hashTable[i].documents,
hashTable[i].documentsCount * sizeof(char));
			for (j = 0; j < hashTable[i].documentsCount; j++)
			{
				//her bi dokuman icin gerekli yer acilip isimler yerlestirilir.
				fscanf(hashFile, "%s", str);
				hashTable[i].documents[j] = (char*)malloc(strlen(str) * sizeof(char));
				strcpy(hashTable[i].documents[j], str);
			}
		}
}

//program kapanmadan once hash dosyasina yazmak icin kullanilan fonksiyon.
void writeToHashFile(WORD_INFO *hashTable, int *currentSize, float *loadFactor)
{
	FILE *fp; //hash dosyasi degiskeni
	int i; //dongu degiskeni
	int j; //dongu degiskeni
	fp = openHashFile("w");
	if (fp == NULL)
	{
		printf("Couldn't open hash file...\n");
		return;
	}
	//dolu hucre sayisi yazilir
	fprintf(fp, "%d ", *currentSize);
	//loadfactor yazilir
	fprintf(fp, "%f ", *loadFactor);
	for (i = 0; i < M; i++)
	{
		//hucre bossa, bos oldugunu bildiren karakter ve dokuman sayisi olarak da 0 yazilir.
		if (strcmp(hashTable[i].word, HASH_TABLE_EMPTY_SLOT_CHAR) == 0)
		{
			//fprintf(fp, "%d ", -1);
			fprintf(fp, "%s ", HASH_TABLE_EMPTY_SLOT_CHAR);
			fprintf(fp, "%d ", 0);
		}
		//hucre bos degilse; kelime, dokuman sayisi ve dokuman isimleri yazilir.
		else
		{
			//fprintf(fp, "%d ", hashTable[i].key);
			fprintf(fp, "%s ", hashTable[i].word);
			fprintf(fp, "%d ", hashTable[i].documentsCount);
			for (j = 0; j < hashTable[i].documentsCount; j++)
			{
				fprintf(fp, "%s ", hashTable[i].documents[j]);
			}
		}
	}
	fclose(fp);
}

//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;
}
```