Dersin adı:Algoritma Analizi

Grup Numarası-2

Ödev-2

Recep Furkan Koçyiğit

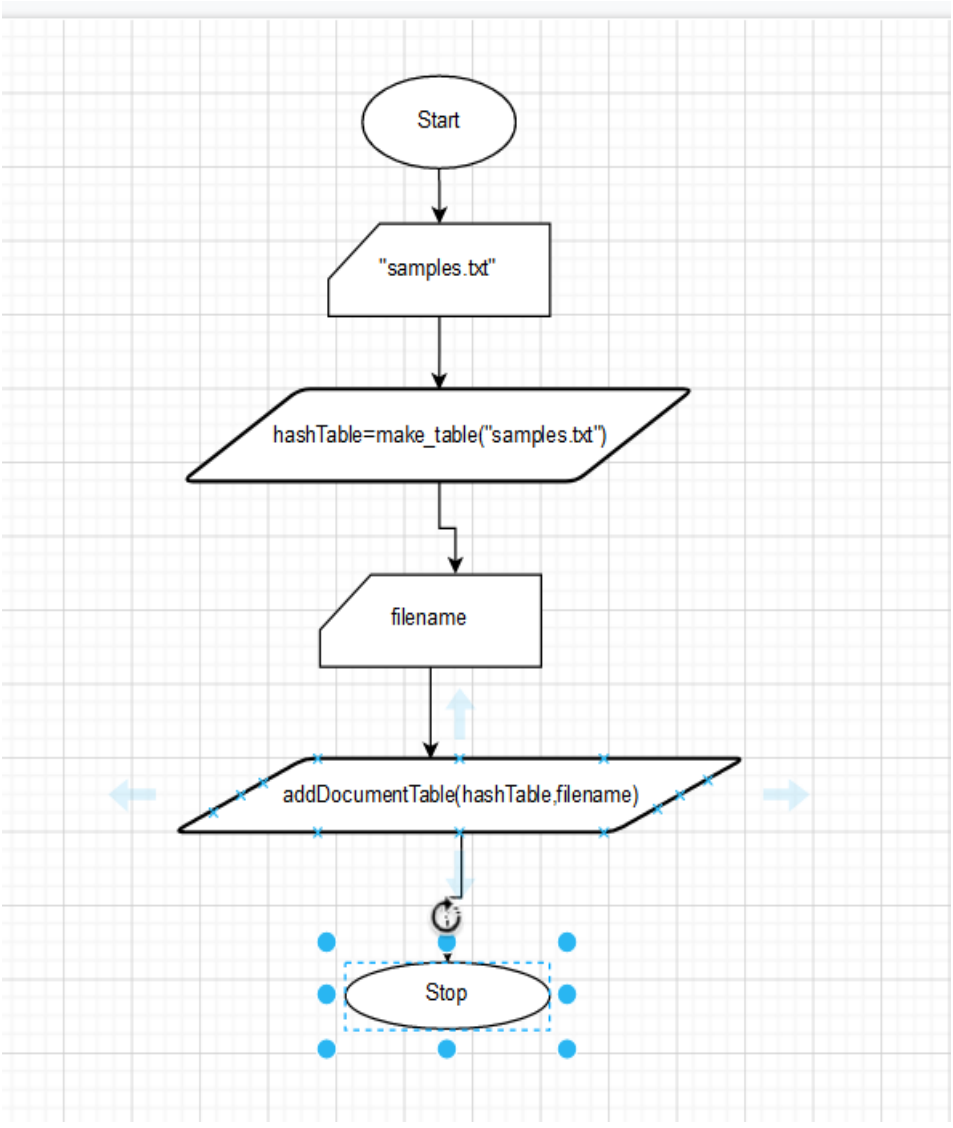
16011043

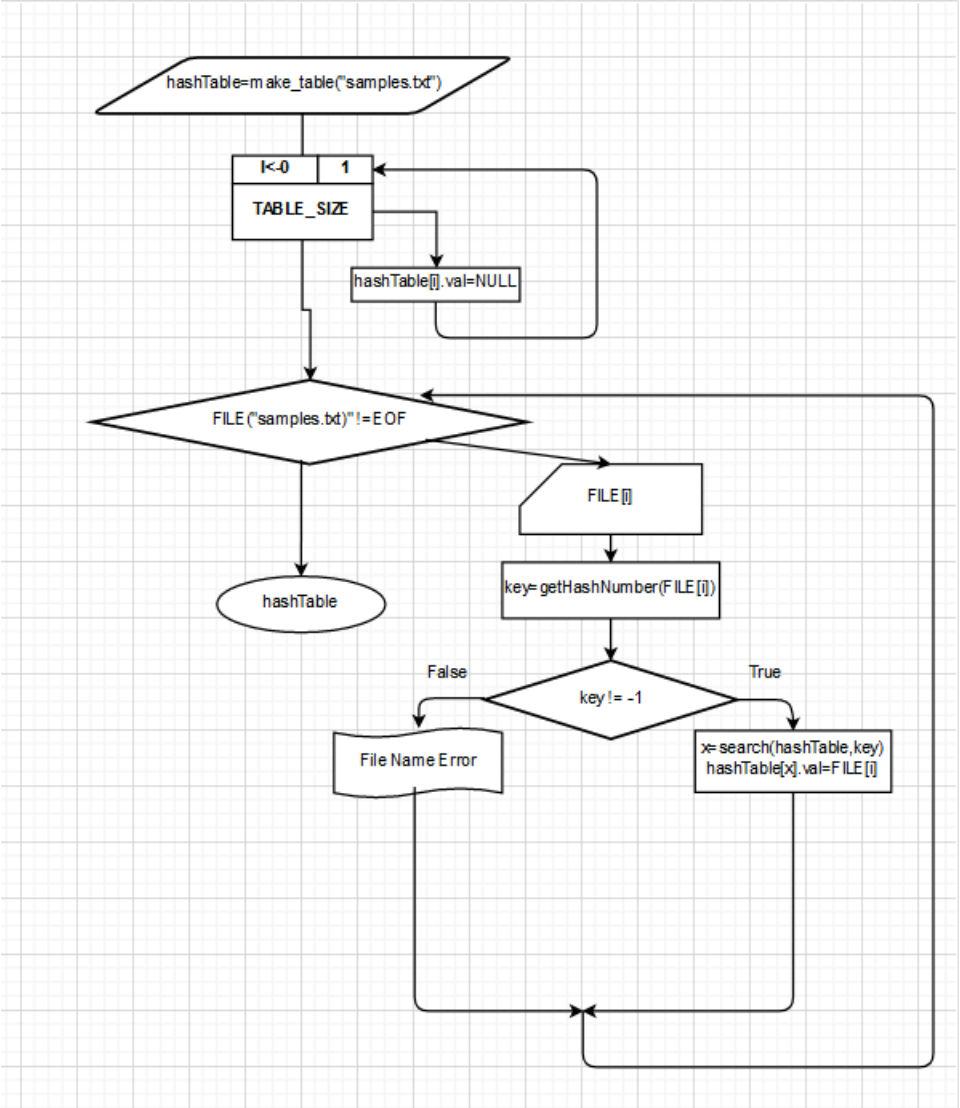
Ödev Konusu:Hashing Algoritmaları

Yöntem:

a-)Problemin Tanımı: Eklenmek istenen yeni bir dokümanın veritabanında olup olmadığını kontrol eden, eğer yoksa veritabanına ekleyen bir sistem tasarlanacaktır. Dokümanın veritabanında olup olmadığının mevcut bütün dokümanların içeriklerine tek tek bakılarak yapılması zaman alıcı bir işlemdir. Bu nedenle bu işlem hashing ile yapılacaktır.

b-)Akış Diyagramı:





Uygulama:

a-)Örnek 1: Küçük ve büyük harflerden oluşan uzun olmayan 10 dosyanın hash adreslerini hesaplayarak hash tablosuna yerleştiriniz. Dosya içeriklerini oluştururken 3 dosya için aynı hash adresini oluşturarak çakışma olmasını sağlayınız. Kolayca çakışma olmasını sağlamak için yukarıda stringi sayıya dönüştürerek key değeri elde ederken kullanılan bağıntıda R=1 alınız.

directory/A.txt	key=91115	Adres=12	
directory/B.txt	key=285268	Adres=8	
directory/C.txt	key=2072	Adres=15	
directory/D.txt	key=64249	Adres=6	
directory/E.txt	key=40373	Adres=15	Yeni Adres=4
directory/F.txt	key=40790	Adres=7	
directory/G.txt	key=56175	Adres=7	Yeni Adres=6 Yeni Adres=5
directory/H.txt	key=71606	Adres=2	
directory/I.txt	key=41861	Adres=7	Yeni Adres=13
directory/J.txt	key=62119	Adres=1	

b-)Örnek 2: Yeni dosya ekleme için bir adet veritabanında olan dosya için, bir adet de veritabanında
olmayan dosya için algoritmanızın çalışmasının ana adımlarına ait değişimi gösteriniz.

Table:

directory/A.txt	key=91115	Adres=12
directory/B.txt	key=285268	Adres=8
directory/C.txt	key=2072	Adres=15
directory/D.txt	key=64249	Adres=6
directory/E.txt	key=40373	Adres=4
directory/F.txt	key=40790	Adres=7
directory/G.txt	key=56175	Adres=5
directory/H.txt	key=71606	Adres=2
directory/I.txt	key=41861	Adres=13

Eklenecek Dosya:

ornek.txt	key=40905	Adres=3

Başarılı şekilde eklendi.

Eklenecek Dosya:

ornek1.txt	key=71606	Adres=2

ornek1.txt dosyasi H.txt ile aynı içereğe sahip,eklenmedi.

Sonuç:

Karmaşıklık Hesabı:

“samples.txt” içinde N farklı dosya olsa ,bu dosyaların en uzunu M kelime içeriyorsa,en uzun kelimenin uzunluğu K ve tablonun uzunluğu T ise:

Node *make_table(char *) işleminin karmaşıklığı = O(N*(N+M*K) +T)

void addDocumentTable(Node *,char *) = O(M*K+N*M)

int search(Node*,lld) = O(N)

void printTable(Node*) = O(T)

int funcH(lld,int) = O(1)

int funcH1(lld) = O(1)

int funcH2(lld) = O(1)

lld getHashNumber(char *) = O(M*K)

lld get_key(char *,int) = O(K)

int compareDocuments(char *,char *) = O(M)

void addFile(char *) = O(M)

Bu probleme ait çözümün karmaşıklığı:

O(N*(N+M*K) +T) dir.

Kaynak Kod:

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

#define lld long long int
#define TABLE_SIZE 17
#define NUMBER_OF_TABLE_ELEMENTS 10

struct node{
        char *val;
};
typedef struct node Node;


Node *make_table(char *);
void addDocumentTable(Node *,char *);
int search(Node*,lld);
void printTable(Node*);
int funcH(lld,int);
int funcH1(lld);
int funcH2(lld);
lld getHashNumber(char *);
lld get_key(char *,int);
int compareDocuments(char *,char *);
void addFile(char *);
```

```
int main(){
	char *filename="samples.txt";//source dosyalarının isimlerinin tutuldugu dosya
	char addDocumentFileName[16];//yeni eklenecek dosyalar icin kullanilmistir
	Node *hashTable=make_table(filename);//source deki dosyalara gore hashtable ın olusturulması
	int x=1;//while dongusunden cikmak icin kullanilmistir
	int option;//hangi operasyonun secildigini okumak icin kullanilmistir
	while(x){
		printf("1-Database goruntule.\n");
		printf("2-Database e dokuman ekle.\n");
		printf("3-Cikis\n");
		printf("Operasyon numarasini giriniz:");
		scanf("%d",&option);
		if(option == 1){
			printTable(hashTable);//hashtable ı yazdırır.
		}
		else if(option == 2){
			printf("Dokumanin adini giriniz:");
			scanf("%s",addDocumentFileName);
			addDocumentTable(hashTable,addDocumentFileName);//table a yeni bir dosya ekler
			printf("\n");
		}
		else if(option == 3){
			x=0;//cikis
		}
		else{
			printf("Lutfen Gecerli bir operasyon giriniz.\n");
			printf("\n");
```

```
		}
	}
	return 0;
}


void addDocumentTable(Node *table,char *filename){//ismi verilen dosyadan key i olusturur table a bakar.Eger key in gosterdigi deger bossa ekler.


	//Degilse ve dosyalar aynı ise eklemez.Dosyalar farklı ise bos yer gorene kadar bu olay devam eder
	lld key=getHashNumber(filename);//Dosyadan keyi elde etme
	if(key == -1){//Dosya acilamadi ise key -1 dir
		printf("Dosya bulunamadi.\n");
		return;
	}
	int i = 0;//double hashing yaparken kacinci seferde bos yerin bulundugunu gosterir
	int index = funcH(key,i);//tabloda nereye yerlestirilecegini gosterir
	int flag = 1;//dosyalarin ayni olup olmadigini kontrol etmek icin kullanilmistir
	printf("%d\n",index);
	while(flag && table[index].val != NULL){//dosyalar ayni degilse ve bakilan index bos degilse doner
		//printf("%s %s\n",table[index].val,filename);
		if(compareDocuments(table[index].val,filename) == 1){//dosyalar ayni ise flag guncellenir ve dongu son bulur.
			flag = 0;
		}
		else{//dosyalar ayni degil ise yeni index elde edilir
			i++;
			printf("%d %d ",index,i);
			index=funcH(key,i);
			printf("%d\n",index);
```

```
                }
        }
        if(flag){//Dosyalar ayni degilse tabloya eklenir.

                char *str1 = "directory/";//eklenecek dizin
                char *newFileName = (char *)malloc(2 + strlen(str1)+ strlen(filename));//hash table a eklemek icin dosyanin adini duzenleme
        strcpy(newFileName, str1);
        strcat(newFileName, filename);
        table[index].val=(char *)malloc(sizeof(char)*strlen(newFileName));
                strcpy(table[index].val,newFileName);//hashtable a dosyayi ekleme

                memset(newFileName,0,sizeof(newFileName));//dosya adini samples.txt ye eklemek icin tekrardan duzenleme
                str1="\ndirectory/";
        strcpy(newFileName, str1);
        strcat(newFileName, filename);

                FILE *fp=fopen("samples.txt","a");
                fputs(newFileName,fp);//samples.txt ye dosyayi yazma
                fclose(fp);

                addFile(filename);//dosyayi directory dizinine ekleme
                printf("File insert successfully.\n");
        }
        else{
                printf("File already in database.\n");
        }
}
```

```
Node *make_table(char *filename){//hash table ı okunan dosya ismine gore olusturur.
	FILE *fp_source=fopen(filename,"r");//kaynak dosya icin kullanilmistir
	char buff_filename[64];//kaynak dosyanin icindeki dosya isimlerini tutmak icin kullanilmistir
	lld key;//okunan dosyanin keyini tutmak icin kullanilmistir.
	int x;//hashtable da eklenecek yerin indisini tutar
	int i;//hashtable uzerinde gezmek icin iterasyon amacli kullanilmistir
	if(fp_source==NULL){//dosya acilamazsa fonksiyon null dondurur
		printf("No Source File");
		return NULL;
	}
	Node *hashTable=(Node *)malloc(sizeof(Node)*TABLE_SIZE);//table ı olusturma
	for(i=0;i<TABLE_SIZE;i++){//table i sifirlama
		hashTable[i].val=NULL;
	}
	while(!feof(fp_source)){//kaynak dosyasını sonuna kadar okuma
		memset(buff_filename,0,sizeof(buff_filename));
		fgets(buff_filename,64,fp_source);//kaynak dosyanin icindeki dosya isimlerini okuma
		if(buff_filename[strlen(buff_filename)-1] == '\n'){
			buff_filename[strlen(buff_filename)-1]='\0';
		}
		printf("%s ",buff_filename);
		key=getHashNumber(buff_filename);//dosyanin keyini olusturma
		if(key != -1){//key olusturulduysa
			x=search(hashTable,key);//keyi tablodaki bos yerin indexi
			hashTable[x].val=(char *)malloc(sizeof(char)*strlen(buff_filename));
			strcpy(hashTable[x].val,buff_filename);//tablodaki bos yere o dosyanin adi yazilir
		}
```

```
                else{
                        printf("File name Error\n");
                }

        }
        fclose(fp_source);//ana dosya kapanir.
        return hashTable;//tablo dondurulur
}
int search(Node *table,lld key){//verilen keye karsilik tablodaki bos yerin indexini dondurur
        int i=0;//kacinci seferde bos yere eklendiginin bilgisini tutar
        int index=funcH(key,i);//ilk sefer icin elde edilen index
        printf("%d ",index);
        while(table[index].val != NULL){//eger bakilan yer bos degilse
                i++;                                    //kacinci seferde eklenecegi bilgisi guncellenir
                printf("|%d| ",i);
                index=funcH(key,i);//yeni index olusturulur
                printf("%d ",index);
        }
        printf("\n");
        return index;//tablodaki bos yerin indexi dondurulur
}
int compareDocuments(char *filename1,char *filename2){
        FILE *fp1=fopen(filename1,"r");
        FILE *fp2=fopen(filename2,"r");
        char ch1;
        char ch2;
        if(fp1 == NULL || fp2 == NULL){
                printf("File did not open!\n");
                return 0;
```

```
	}
	while(!feof(fp1) && !feof(fp2)){
		fscanf(fp1,"%c",&ch1);
		fscanf(fp2,"%c",&ch2);

		if(ch1 != ch2){
			fclose(fp1);
			fclose(fp2);
			return 0;
		}
	}
	fclose(fp1);
	fclose(fp2);
	return 1;
}
void addFile(char *filename){//adi verilen dosyayi dizene kaydeder
	char *str1 = "directory/";//dosyanin kaydedilecegi yol
	char *newFileName = (char *)malloc(1 + strlen(str1)+ strlen(filename));//kaydedilecek dosya adi guncellenir
   strcpy(newFileName, str1);
   strcat(newFileName, filename);
	char ch;//dosya char char kaydedilecektir.bunun icin kullanilmistir
	FILE *fp1=fopen(newFileName,"w");//kaydedilecek dosya
	FILE *fp2=fopen(filename,"r");//okunan dosya
	if(fp1 == NULL){
		printf("New file did not create.\n");
		fclose(fp2);
		return;
	}
```

```
	while(!feof(fp2)){//dosya kayit islemi
		fscanf(fp2,"%c",&ch);
		fprintf(fp1,"%c",ch);
	}
	fclose(fp1);
	fclose(fp2);
}
lld get_key(char *word,int n){//okunan kelimeye ait key degerini dondurur
	int i;//n uzunluktaki kelime icin iterasyon amacli kullanilmistir
	int R=1;//formuldeki R sayisi
	lld key=0;//keyin baslangic daki durumu
	for(i=0;i<n;i++){//keyin bulunmasi
		key+=word[i] * (int)pow((double)R,(double)n-i-1);
	}
	return key;
}
void printTable(Node *table){//hash tablosunu yazdirir
	int i=0;
	printf("Database Table:\n");
	for(i=0;i<TABLE_SIZE;i++){
		printf("%d %s\n",i,table[i].val);
	}
	printf("\n");
}
lld getHashNumber(char *filename){//verilen dosya ismini okur ve ona ait key degerini uretir
	char word[64];//dosyanin icinde gecen kelimeleri (max 64 karakter) okumak icin
kullanilmistir
	char ch;//dosya karakter karakter okunacaktir.bu yuzden kullanilmistir
	char bch;//dosyada kendinden once okunan karakterin bilgisini tutar
```

```
	lld key=0;//key basta 0 dir
	int j=0;//her bir kelimenin uzunlugunu tutar
	FILE *fp_assitant=fopen(filename,"r");

	if(fp_assitant == NULL){//dosya acilamazsa
		return -1;
	}
	while(!feof(fp_assitant)){
		while(fscanf(fp_assitant,"%c",&ch) != EOF){//karakter okuma
			if((ch <= 'Z' && ch >= 'A') || (ch <= 'z' && ch >= 'a')){//karakter alfabetik
ise word e eklenir
				word[j++]=ch;
				word[j]='\0';
			}
			if((ch == ' ') || ((ch == '\n') && (bch == '.'))){//karakter bosluk ise veya
dosya sonu geldiyse
				key+=get_key(word,strlen(word));//suana kadar okunan
kelimeye ait key

				if(ch == ' ')//bosluk varsa key e eklenir
					key+=' ';
				j=0;
			}
			bch=ch;//onceki karakter bilgisi guncellenir
		}
	}
	fclose(fp_assitant);
	printf("key=%lld ",key);
	return key;
}
```

```
int funcH(lld key,int i){//hash fonksiyonu verilen key in tablodaki indexini dondurur
	return (funcH1(key) + i * funcH2(key)) % TABLE_SIZE;
}
int funcH1(lld key){//hash1 fonksiyonu double hashing icin gerekli ilk fonksiyon
	return key % TABLE_SIZE;
}
int funcH2(lld key){//hash2 fonksiyonu double hashing icin gerekli ikinci fonksiyon
	return 1 + (key % (TABLE_SIZE - 1));
}
```