Algoritma Analizi Dönem Projesi

ELİF YAĞMUR DURAN

18011071

Contents

Tanımlar

#define M1 97 #define M2 96 #define R 31	Preprocessor için tanımlar. M1 boyutu, M' doublehash leme işleminde kullanılacak M1-1 i temsil eder. R horner için asal sayıdır.

struct person_info {
char id[5];
int key;
char name[50];
char surname[50];
int year;
char city[25];
bool deleted;
};
struct person_info *hash_table[M1];
	Bilgilerin doldurulacağı struct tır. Daha sonra global olarak bir struct array i tanımlanır.
	İd kısmı key üretiminde kullanılacaktır, bir kez hesaplandıktan sonra key struct a ilerde kullanılmak için kaydedilir, deleted kısmı ise delete fonksiyonunda kullanılacaktır.

int i, j, k;
int size;
int item_count = 0;
float loadfactor;
// file temps
char filestr[250];
struct person_info temp_item;
FILE *fptr;
	Kullanışlılığı artıracak main içi değişkenler. Sayaç için, file almak okumak için, konulan eleman sayısı takibi için.

Fonksiyonlar

Hashleme Fonksiyonları

void print_hashfunc() {
printf("current hashtable look is:");
int i;
for (i = 0; i < M1; i++) {
	İşlem tamamlandıktan sonra tabloyu bastırmak içindir.

if (hash_table[i]->key != 0) {
printf("\ni = %d | id: %s / Name, Surname: %s %s / Year: %d / City: %s", i, hash_table[i]->id, hash_table[i]->name, hash_table[i]->surname, hash_table[i]->year, hash_table[i]->city);
}
}
}	

int horners(char str[5]) {
int hashed_key;
int i;
for (i = 0; i < 5; i++) {
hashed_key = R * hashed_key + (str[i] - 'A' + 1);
}
return hashed_key;
}
	String şeklinde id alıp horner metodu uygular. Verilen formül üzerinden her string karakterini işleme sokar ve integer tipinde bir key döndürür.

int h1(int key) {
return key % M1;
}
int h2(int key) {
return 1 + (key % M2);
}
int double_hash(int key, int i) {
return (h1(key) + (i * h2(key))) % M1;
}
	Dokümanda verilen double hash fonksiyonunu işleyen fonksiyonlar.

Find

struct person_info *find_hashfunc(char id[5]) {
int key = horners(id);
int i = 1;
int index = double_hash(key, i);
 	
while ((hash_table[index]->key != 0) && i <= M1) {
if (hash_table[index]->key == key) {
return hash_table[index];
}
i++;
index = double_hash(key, i);
}
return NULL;
}
	Verilen id deki struct ın hash tablosunda nerede olduğunu bulan fonksiyon. Öncelikle bu id nin key ini bulur, daha sonra da bu key üzerinden gerekli index i üretir.
	Hash tablosunu temsil eden struct array inde bulunan indexten başlayarak, boş eleman olmadıkça (bu key in 0 olmasıyla temsil edilir) ya da tablodan çıkılmadıkça (<= M1) her eleman tek tek aranır. Döngü içerisinde aranan bulunduğu anda o indexteki struct döndürülür. Bulunamazsa o iterasyon için i artılıp index bu i ye göre yeniden doublehash formülü ile hesaplanır ve aranan elemanın olabileceği diğer seçenek indexe gidilir. Bu döngüden çıkıldığında tablo bitmiş ise eleman bulunamamıştır, NULL döndürülür.

Insert

void insert_hashfunc(char id[5], struct person_info temp) {
  struct person_info *item = find_hashfunc(id);
  int key = horners(id);
  int i = 1;
  int index = double_hash(key, i);
 	
  item = (struct person_info *)malloc(sizeof(struct person_info));
 	
  strcpy(item->id, temp.id);
  strcpy(item->name, temp.name);
  strcpy(item->surname, temp.surname);
  item->year = temp.year;
  strcpy(item->city, temp.city);
  item->deleted = false;
	Hash tablosuna eleman ekleme fonksiyonudur. Öncelikle double hash yöntemi ile elemanın olması hereken yer bulunur. Dosyadan okunanları alan geçici bir person_info struct ı açılır ve bilgiler buraya alınır.

while (hash_table[index] != NULL && i <= M1) {
  index = double_hash(key, i);
  i++;
}
 	
if (i == M1) {
  return;
} else {
  hash_table[index] = item;
}
}
	Open adressing uygulamasına göre eğer formülden çıkan index doluysa formül boş yer bulunana kadar yeniden çalıştırılıp yeni index üretilmelidir.
	Hash indexi null olmadığı sürece ve tablo sınırları aşılmadığı sürece boş yer arayan loop a girilir. Boş yer bulunamazsa o iterasyon için i artılıp index bu i ye göre yeniden doublehash formülü ile hesaplanır ve diğer seçenek indexe gidilir. Bu döngüden çıkıldığında i M1 i yani tablo boyutunu aşmışsa yer bulunamamıştır geri dönülür. Ancak aşmamışsa son kalınan yere temp struct ındaki bilgiler atanır.

Delete

void delete_hashfunc(char *id) {
  struct person_info *item = find_hashfunc(id);
  if (item == NULL) {
  return;
  } else {
  strcpy(item->id, "0");
  strcpy(item->name, "0");
  strcpy(item->surname, "0");
  item->year = 0;
  strcpy(item->city, "0");
	Hash tablosundan eleman silme fonksiyonu.
	Verilen id ye göre find fonskiyonunu çalıştırır. Eğer sonuç tabloda olmadığı ise fonksiyondan dönülür. Ancak eğer bulunursa her eleman 0 a atanır ve deleted özelliği tru olarak update edilir.

item->deleted = true;
return;
}
}	

Main

fptr = fopen("test.txt", "r");
if (fptr == NULL) {
printf("test.txt failed to open.");
exit(0);
} else {
printf("\nThe file is now opened.\n");
while (fgets(filestr, 50, fptr) != NULL) {
sscanf(filestr, "%s %s %s %d %s", temp_item.id, temp_item.name, temp_item.surname, &temp_item.year, temp_item.city);
insert_hashfunc(temp_item.id, temp_item);
item_count++;
}
fclose(fptr);
printf("\n\nThe file is now closed.\n");
print_hashfunc();
}
return 0;
	File ın açılıp açılmadığının kontrolü yapılır. Daha sonra her okunan satır bir temp structına alınır, bu struct ilerde insert fonksiyonuna gönderilecektir., item_count ta artırılır. File okunması bitirilince file kapatılır.