TRAKYA ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

C DİLİNDE KULLANILAN VERİ TABANLARI VE PERFORMANSLARI

NİLGÜN TOSUN

YÜKSEK LİSANS TEZİ

BİLGİSAYAR MÜHENDİSLİĞİ ANABİLİM DALI

Bu tez 19./02/1998 tarihinde aşağıdaki jüri tarafından kabul edilmiştir.

(imza)	(imza)	(imza)
Yrd. Doç. Dr.	Yrd. Doç. Dr.	Yrd. Doç. Dr.
Nurşen SUÇSUZ	Cavit TEZCAN	Ercan BULUŞ
Danışman		

TRAKYA ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

C Dilinde Kullanılan Veri Tabanları ve Performansları

Yüksek Lisans Tezi

Nilgün TOSUN
Tez Yöneticisi
Yrd. Doç. Dr. Nurşen SUÇSUZ

Edirne, 1998

İÇİNDEKİLER

ÖZET

Bu çalışmada , Btrieve ve Db-Vista veri tabanları ile kayıt işlemlerinin C fonksiyonlarıyla IPR türü ikili ağaç üzerinde elle yapılması arasında performans analizi yapılmış; elde edilen sonuçlar grafikler ve tablolarla kıyaslanmıştır.

Tezin tamamı beş bölümden oluşmaktadır. Birinci bölüm, yukarıda adı geçen veri tabanları ile IPR türü ikili ağaçlar hakkında genel bilgileri içermektedir.İkinci bölümde sırasıyla, tez programlarının yazılmasında kullanılan bilgisayar, programlarda kullanılan kayıt deseni, Btrieve ve Db-Vista'ya ait dosyalama fonksiyonları ile bu fonksiyonlara ilişkin örnekler açıklamalarıyla verilmiştir. Tezin üçüncü bölümünde ise, performans analizi için yazılan C programlarının akış diyagramları ve dökümleri yeralmaktadır. Dördüncü bölüm, programlarda elde edilen sayısal verilerin tablo ve grafiklerle ifadesini içermektedir. Son olarak beşinci bölümde ise programlardan elde edilen verilerin değerlendirmesi ve tez üzerinde yapılabilecek yeni çalışma önerileri bulunmaktadır.

SUMMARY

In this study, an analysis between Btrieve, Db-Vista Database recording processes and realizing the same process by hand with C funtions on an IPR kind Binary Tree is done; and the results obtained from the analysis are compared by diagrams and tables.

This thesis consist of five chapters. The first chapter includes general information about databases and IPR kind Binary Trees which are mentioned above. The second chapter in an arranged order contains the computer that is used in writing theses, the record pattern that is used in programs, the Filing Functions about Btrieve and Db-Vista and their exemplars with their explanations are given.

The third chapter contains the flow charts and their output written for the performance analysis of the C programs.

The fourth chapter includes the explanation of the numerical data that is obtained from programs with the help of diagrams and tables.

Finally, the fifth chapter, there are suggestions on new studies which can be done on this thesis.

GİRİŞ

C dili 3. jenerasyon bir programlama dili olmakla beraber, Assembler diline alternatif olarak geliştirilmiş B dilinin bir sonraki gelişmiş halidir. Bu nedenle, C programlama dili büyük oranda sistem programlama dili olarak kullanılmaktadır. Yani 4. jenerasyon bir database (örneğin foxpro veya bir çok veri tabanı dili) veya herhangi bir işletim sistemi C dili ile yazılabilir.

C programlama dili genel amaçlı bir dildir. Yani C dili ile herhangi bir programlama işlemi yapılabilir. Herhangi bir bilimsel hesaplama, mühendislik işlemleri, işletim sistemi yazma, 4. jenerasyon bir dil yazma gibi her türlü amaca yönelik yazılım geliştirmeyi C ile yapabiliriz. Bir programlama dili, bünyesinde veri tanımlama (DDL), veri işleme dili (DML), veri kontrol komutlarını içermez. Eğer bunları içerseydi bu dile programlama dili denmezdi. Çünkü içerisinde veri işleme işlemlerini bulundurduğundan dolayı veri tabanı dili olurdu. Veri tabanı diline örnek olarak veri tanımlama (DDL), veri işleme (DML), veri kontrol dili komutlarını içeren 4. jenerasyon herhangi bir veri tabanını gösterebiliriz. Bir programlama dili, desteklemiş olduğu veri yapılarının (float, structure) diske yazılmasını ve diskten okunmasını sağlayan yüksek seviyeli giriş/çıkış fonksiyonlarını desteklemek zorundadır. Bu nedenle günümüzde veri işleme işlemleri bu fonksiyonlarla yerine getirilemeyeceğinden dolayı, veri tabanları ve veri yönetim sistemlerinin kullanılması gerekmektedir.

Veri tabanı, çeşitli amaçlar için düzenli olarak veri saklayan ve isteğe göre bilgi üreten bir kayıt tutma sistemidir. Veri tabanı bir sistem halinde düzenlenmiş veri topluluğudur ve sistem kullanıcılarının hepsi, aynı veri tabanından kendileri için gerekli olan veriye her an erişebilirler. Veri tabanı sistemleri, kişi ve kuruluşların özel ihtiyaçlarına göre çeşitlilik gösterir.

Veri tabanlarının başarısı veri paylaşımı, bağımsızlık, tutarlılık ve veri bütünüğünü desteklemesindn dolayıdır. Gelneksel dosya yönetim sistemleri bu özellikleri doğal olarak desteklemezler.

Veri tabanları donanım kazalarından, disk ortamı bozukluklarından ve bazı kullanıcı hatalarından korunduğu için hata kurtarma özelliğine sahiptir. Çoklu kullanıcılar ve çoklu uygulama programları aynı veri tabanına veri yazıp, bu veri tabanlarından veri okuyabilirler. Veriler yetki verilmemiş okuma-yazma işlemlerine karşı korunabilmektedir.

Yukarıda kısaca bahsedilen özellik ve avantajlarından dolayı, 60'lı yıllardan beri veri tabanı sistemleri çeşitli amaçlar için kullanılagelmektedir.

TEŞEKKÜR

Bu çalışmanın planlanması, gerçekleştirilmesi ve değerlendirilmesi sırasında yakın ilgi ve özveriyi esirgemeyen danışman hocam Yrd. Doç .Dr. Nurşen SUÇSUZ'a, tezimin başlangıç aşamasında fikirlerinden yararlandığım hocam Yrd. Doç. Dr. Cavit TEZCAN'a, kaynak araştırmalarım sırasında bana yardımcı olan hocalarım Yrd. Doç. Dr. Ercan BULUŞ ve Yrd. Doç.Dr. Servet ŞENYÜCEL'e , arkadaşım Arş. Gör. Seyhan BASMACI'ya ve benden desteklerini esirgemeyen aileme teşekkür ederim.

1. DATABASE' LERİN TANITIMI

1.1. BTRIEVE RECORD MANAGER (BTRIEVE KAYIT YÖNETİCİSİ)

Btrieve, kullanıcıya birden fazla kayıtı kolayca yönetme, bu kayıtları birden fazla anahtara göre indexleme ve bu kayıtlar üzerinde yapılacak her türlü işlemi kısa zamanda gerçekleştirme olanağı sağlayan bir kayıt yönetim sistemidir. Btrieve ile kayıtlar üzerinde yapılabilen temel işlemler kayıta ulaşma, kayıt üzerinde düzeltme, kayıt silme ve update gibi klasik dosyalama işlemlerdir.

Btrieve'in çalıştırılabilmesi için en az IBM 386/486 tabanlı bir bilgisayara ihtiyaç vardır. Btrieve kayıt yönetim sistemi DOS 3.x ve üzeri, OS/2, Windows 3.1, Windows NT ve Windows 95 gibi işletim sistemleri altında çalışabilme özelliğine de sahiptir. Btrieve' in bir başka özelliği ise değişik programlama dillerini desteklemesidir. Bu diller şöyle sıralanabilir :

- Microsoft Quick Basic
- IBM Basic
- IBM Pascal
- Turbo Pascal
- Microsoft Cobol
- MS C
- C ++
- Visual Basic
- Visual Pascal
- Visual Cobol

1.1.1. BTRIEVE' in TEMEL ÖZELLİKLERİ

1-Kayıtlar için çok anahtarlı erişime izin verir.

2-Dosyalar arasında ilişkisel erişim mevcuttur.

3-Tüm anahtarların otomatik bakımı, Btrieve tarafından yapılır.

4-Duplicate, modifiable, segmented, null ve descending türlerinde değişik anahtar desteği vardır.

5-İlave index'lere izin verir.

6-Extended anahtarlar dışında 13 farklı extended anahtar tipini kullanır.

7-Lotus/Intel/Microsoft Expanded Bellek Yöneticisi özelliğine sahiptir.

8-Bölünmüş dosyalar kullanır.

9-I/O buffer cache.

10-Doğruluk kontrolleri yapılır.

11-Bağımsız dosya büyüklükleri kullanılabilir.

12-Network ve Multitasking data erişim kapasitesi mevcuttur.

13-Dosyaların bakımı ve onarılması için gerekli tüm utilitiy setine sahiptir.

Yukarıda maddeler halinde sıralanan özellikleri kısaca açıklamak gerekirse :

- **Çok Anahtarlı Erişim**

Btrieve' de dosyaların herbirine 24 farklı yoldan ulaşabilmek mümkündür. Yani, 24 anahtar sahası ya da erişim yolu (access path) denilen alanlar tanımlanabilir. Btrieve, tanımlanan herbir erişim yolu için ayrılmış bir dahili index tutar. Indexlerde bu kayıtlara, isteğe göre, azalan ya da artan sırada erişme imkanı vardır.

- **Anahtarların Otomatik Bakımı**

Btrieve anahtar değerlerine dayanan kayıtlar için tüm index'leri otomatik olarak update eder (günceller). Kullanılan uygulama programına yeni kayıtlar eklediğinde,

Btrieve index'e yeni anahtar girişlerini ilave eder. Kullanılan uygulama programı kayıtları sildiğinde ise, Btrieve de index girişlerini siler. Btrieve aynı zamanda update edilen (güncellenen) kayıtlar için de index'ler üzerinde düzenlemeler yapar.

- **Duplicate, Modifiable, Segmented, Null ve Descending Anahtarlar**

Anahtarlar, bir dosyadaki kayıtları belirlemek için kullanılan alanlardır.Btrieve, bir anahtarı kullanarak, bir dosyadaki tüm kayıtlar içinden isteneni doğru olarak seçebilir. Btrieve' de bir dosya oluşturulduğunda, dosyadaki her bir anahtar için 5 farklı özellik tanımlanabilir. Bu beş anahtar özelliği: Duplicate, modifiable, segmented, null value ve descending order.

DUPLICATE

Bu özellik, dosyada birden fazla kayıtın aynı anahtar değerini taşımasına izin verir. Eğer bir anahtarın "duplicate" özellik taşıması istenmiyorsa, dosya içindeki her kayıt kendi anahtar değeri ile tek olacak şekilde tanımlanır.

Btrieve, duplicate anahtarları, dosyaya girişin kronolojik sırasında saklar. Eğer özellikle bir anahtarın duplicate olmaması belirtildiyse, Btrieve, anahtar alanında aynı değeri taşıyan birden çok kayıtın dosyaya eklenmesine izin vermez.

MODIFIABLE

Btrieve'de çalışırken bir anahtar "MODIFIABLE" olarak da tanımlanabilir. Eğer bir anahtar modifiable ise, Btrieve, kullanılan uygulama programına, varolan bir kayıtı update etme ve anahtar alanının içeriğini değiştirme iznini vermiş demektir. Örneğin, eğer hesap farkı anahtar alanlardan biri ise, müşteri alış-veriş yaptığı ve borç

ödediği için, uygulama programına bu alanın değerini değiştimesi için izin verilebilir. Bununla birlikte, hesap numarası bilgisini kullanarak, non-modifiable anahtar özelliğinde bir anahtar alan oluşturmak da mümkün. Çünkü, müşterinin hesap numarası asla değişmeyecektir.

SEGMENTED

Anahtarlar, her bir kayıttaki bir ya da daha çok segmenti kapsar. Bir segment genel olarak kayıttaki bir sahaya karşılık gelir ve segmentler birbirini takip edecek şekilde yanyana sıralı değildir. Bir anahtarın toplam uzunluğu, anahtar segmentlerinin uzunlukları toplamına eşittir. Toplam uzunluğun maximum değeri 255 karakterdir. Anahtar segmentlerinin maximum sayısı, sayfa boyutuna bağlı olarak belirlenir. Eğer sayfa boyutu 512 Byte ise, en fazla 8 segment tanımı yapılabilir.Bir sayfa boyutu 1024 Byte veya üzeri olan bir dosya için, en fazla 24 anahtar segmenti tanımlanabilir. Bir sayfasının boyutu 512 Byte veya üzeri olan bir dosya, 8 segmentli bir anahtar ya da her birinde bir segment olan 8 anahtar ya da bunların kombinasyonundan oluşan bir yapıyı içerir.Eğer bir dosyanın sayfa boyutu 1024 Byte veya üzerinde bir değer ise, 24 segmentli bir anahtar ya da bir segmentli 24 anahtar ya da bunlar arasında bir kombinasyondan oluşan bir yapıyı kapsar.

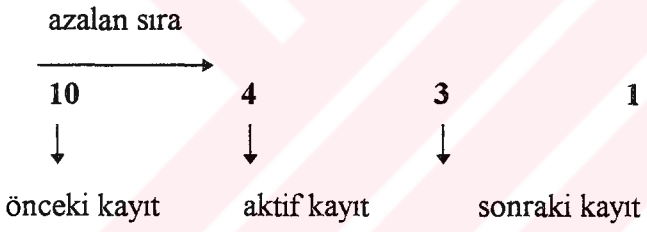
NULL

Btrieve'de bir anahtar için "null" değer tanımı da yapılabilir. Çok genel olarak null değerler, ASCII blank (hex. 20) ve binary 0 ile ifade edilir. Eğer bir anahtar sahasındaki herbir byte null değer içerirse, bir kayıt ilavesi yapıldığında bu anahtarın index'i update edilmez. Btrieve'in bir anahtar yardımıyla dosyayı sıralı olarak okuması sırasında, null değer içeren kayıtlar bulunmaz. Bunun yerine, erişim yolundaki, anahtar alanında null değer ihtiva etmeyen bir sonraki kayıta ulaşılır.

DESCENDING ORDER

Btrieve normalde anahtar değerlerini küçükten büyüğe doğru artan şekilde sıralar. Eğer istenirse bir anahtar segmenti tanımlanırken, Btrieve'e, büyükten küçüğe doğru bir sıralama yapılmak istendiği özellikle belirtilebilir.

Btrieve işlem sonuçlarını, index'teki anahtar değerlerinin sırasına bağlı olarak üretir. Yani, anahtar değerlerinin karşılaştırılması ile işlem yapılmaz. Örneğin Get Greater (aktif kayıtın büyüğünü getir) işlemini bir azalan sıralı anahtarda yaptırırsak, Btrieve anahtar buffer'ında belirtilen anahtar değerinden daha küçük anahtar değerli ilk kayıta erişir.Aynı şekilde, bir Get Less Than (aktif kayıtın küçüğünü getir) işlemini azalan anahtar kullanarak yaptırırsak. Btrieve, anahtar buffer'ında belirtilen değerden daha büyük anahtar değerli kayıta erişir.



- **İlave Index'ler**

Btrieve, dosya oluşturulduktan sonra, kullanıcıya o dosya için değişik indexler tanımlamasına izin verir. Bu indexlere SUPPLEMENTAL (ilave) INDEX adı verilir. Kullanıcı bir supplemental index oluşturduğunda Btrieve, dosyadaki değişen dataların bakımını otomatik olarak yapar.

- **Extended Anahtar Tipleri**

Btrieve iki anahtar tipini destekler: **Standart** ve **Extended**. Kullanılan standart anahtar tipleri String ve Binary'dir. Standart String anahtarlar ASCII olarak yönetilir. Standart Binary anahtarlar ise işaretsiz 2-Byte'lık integerler olarak düzenlenmiştir.

Extended anahtarlar, genel olarak kullanılan 13 data tipinin dahili saklama formatı tabanlı anahtar değerlerini Btrieve'in tanıyıp karşılaştırmasına izin verir. Bu özellik, Btrieve dosyasının indexlerinin dizaynındaki en büyük esnekliktir.

Standart anahtarlar ya Binary ya da String tiptedir. Binary anahtarlar işaretsiz tamsayılar olarak kullanılıp sağdan sola doğru karşılaştırılırken, string anahtarlar soldan sağa doğru ve byte byte karşılaştırılır.

Extended anahtarlar olarak bilinen 13 anahtar tipi ve kodları şunlardır :

anahtarın tipi	kodu
string	0
integer	1
float	2
date	3
time	4
decimal	5
money	6
logical	7
numeric	8
bfloat	9
lstring	10
zstring	11
unsigned binary	12

string

Btrieve'de bir string tip, soldan sağa doğru dizilmiş karakterlerin bir sırasıdır. Her karakter tekli bir byte'ta ASCII formatında gösterilir.

lstring

Bu bir Pascal stringine karşılık gelir. Düzenli bir stringin tüm karakteristiklerine sahiptir. String'in ilk byte'ı, string uzunluğunun binary gösterimini içerir. Lstring'in 0. Byte'ında saklanan uzunluk, önemli Byte'ların sayısını gösterir. Btrieve, string'in belirtilen uzunluğu ötesindeki herhangi değeri atlar.

zstring

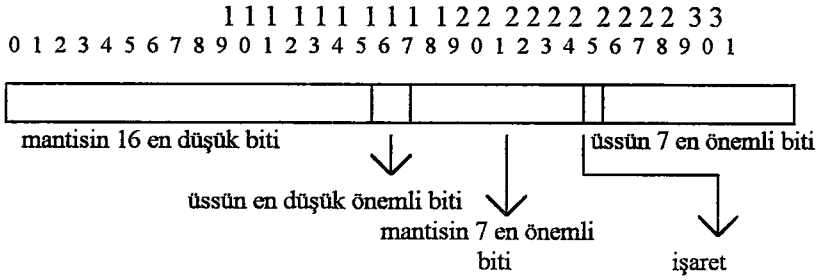
Bu değişken C dilindeki string değişkene karşılık gelir.

Integer

Integer'lar, Intel binary integer formatında saklanırlar. En yüksek ve en düşük byte'lar, bir word'de gösterilirler. Btrieve anahtarı sağdan sola doğru değerlendirir.İşaret, en düşük byte'ın ilk parçasında saklanır. Bu yerleşim, pek çok dilin integer saklama formatı ile uyumludur.

float

4 Byte'lık bir format, 23 bit mantis, 8 bit üs, 1 işaret biti.

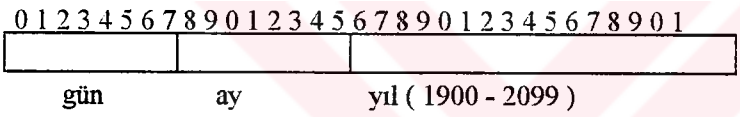


Şekil-1 Float tipi anahtar deseni

Float sayılar 8 Byte'lık da olabilir. Bu şekilde format: 52 bit mantis, 11 bit üs, 1 işaret biti.

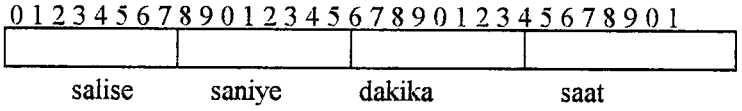
date

4 Byte'lık bir değişkendir.



Şekil-2 Date tipi anahtar deseni

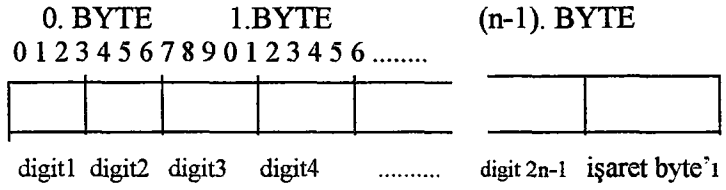
time



Şekil-3 Time tipi anahtar deseni

decimal

Bir n Byte'lık decimal sahanın gösterimi :



Şekil-4 Decimal tipi anahtar deseni

money

Decimal gösterimle aynıdır.

logical

1 ya da 2 Byte değer olarak saklanır. Btrieve mantıksal anahtarları bir string gibi okur ve öyle karşılaştırır. True ya da False olacak şekilde gösterilir.

bfloat

Microsoft BASIC ile uyumlu bir formatta, single ya da double uygun real sayı olarak depolanır. Single2da 32 bit mantis, 8 bit üs, 1 işaret biti. Double'da ise 55 bit mantis, ilk 4 Byte'ta en düşük önemli 32 bit saklanır.

numeric

Numeric değerler, ASCII stringler olarak depolanır.

unsigned binary

Btrieve, bu anahtarları işaretsiz integer gibi arar. Yani sağdan sola doğru karşılaştırma yapılır.

1.1.2. BTRIEVE' DE MANTIKSAL DOSYA KAVRAMLARI

Dosya: Birbiriyle ilişkili kayıtlar topluluğu.

Kayıt: Herhangi bir konuda ardarda gelen bir ya da daha fazla sayıdaki bilgi kalemleridir.

Anahtar: Bir kayıtın tanıtılması ya da konumlandırılması için kullanılan bir karakter setidir.

Index: Bir dosyadaki kayıtlarla bağlantılı anahtar değerlerinin sıralanmış bir setidir.

- DOSYA -

Bir dosya, Btrieve'i kullanarak erişebildiğimiz en yüksek seviyeli database varlığıdır. Btrieve'de bir dosyanın maximum büyüklüğü 4 GB'tır.

- KAYIT -

Kayıt, bir Btrieve dosyasındaki birbiriyle bağlantılı mantıksal data kalemlerinin bir setini gösterir. Dosyalarda kullanılacak kayıt sayısında herhangi bir sınırlama yapılmamıştır. Btrieve'de bir kayıt fixed length (sabit uzunluklu) ya da değişken uzunlukta bir parçayı izleyen bir sabit uzunluklu parçadan oluşur. Dosya için tanımlanan her anahtar, kayıtın sabit uzunluktaki kısmına yerleşir. Bir kayıtın sabit kısmının maximum uzunluğu, dosya için tanımlanan fiziksel sayfa boyuna ve dosya için tanımlanan duplicate anahtarların sayısına bağlıdır. Btrieve, bir kayıtın sabit uzunluktaki kısmının 4090 Byte kapsayabilmesine imkan sağlar. Değişken uzunlukta, 64K'lık kayıtlar tanımlanabilir.

Bir Btrieve dosyası oluşturulurken, dosyada bulunması istenen değişken uzunluktaki kayıtlar özellikle belirtilir. Çünkü dosyanın her kayıtı aynı uzunlukta olmaz.

Kullanıcı, değişken uzunlukta kayıtlardan oluşan bir dosya oluşturduğunda, bu kayıtların sabit uzunluklu kısımları için bir değer tesbit edilmelidir.

- ANAHTAR -

Bir dosyadaki kayıtların belirlenmesi için kullanılan alanlar Anahtar olarak adlandırılır. Btrieve'de kullanılan anahtar tipleri daha önce açıklanmıştı. Bu anahtarlar sayesinde Btrieve, bir dosyadaki tüm kayıtlar içinden isteneni doğru olarak seçer. Btrieve'de bir anahtarın maximum uzunluğu 255 Byte'tır. Dosya başına düşen anahtar sayısı ise 119 olarak belirlenmiştir.

- INDEX -

Bir index, anahtar değerlerini içeren bir Btrieve yapısıdır. Btrieve dengelenmiş B-tree yapısındaki indexleri dinamik olarak tutar. Bir kayıt insert, update ya da delete işlemine tabi tutulduğunda, Btrieve en son değişiklikleri yansıtmak için tüm indexleri düzenler.

Kullanıcı bir dosya yarattığında, Btrieve'in bir index oluşturmak için yararlanması amacıyla en azından bir anahtar tanımı yapmalıdır. Dosya oluşturulurken yapılan bu index PERMANENT INDEX olarak adlandırılır. Dosya oluşturulduktan sonra da istenen değişkene göre index oluşturulabilir. Sonradan oluşturulan bu index'ler SUPPLEMENTAL INDEX ismini alır. Bir dosya için toplam permanent ve supplemental index sayısı 24'ü aşmamalıdır.

Her iki index tipinin duplicate anahtarları saklama ve karşılaştırma şekli birbirinden farklıdır. Bir permanent index'te Btrieve, duplicate değerleri kronolojik sırasında karşılaştırır. Bir supplemental index 'te ise anahtar değerleri buraya fiziksel sıralanışları ile kayıt edilir.

1.1.3. BTRIEVE'DE FİZİKSEL DOSYA KARAKTERİSTİKLERİ

1-Dosya Kontrol Kayıtı: Her dosya, ilk sayfasında bir FCR (File Control Record) bulundurur. Bu sayfada, dosyanın büyüklüğünü ve dosya ile ilgili diğer karakteristikleri tutar.

2-Data Pages: Herbir data page, bir ya da daha çok sayıda kayıt içerir. Sayfa başına düşen data kayıtlarının sayısı, dosya ilk oluşturulduğunda tanımlanan kayıt uzunluğuna göre değişir.

3-Index Pages: Data kayıtlarına erişmek için gereken anahtar değerlerini içerir. Sayfadaki her anahtar değeri, bir kayıt adresine sahiptir. (Eğer duplicate anahtar belirtilmiş ise 2 adres)

4-B-trees: Btrieve, B-trees formundaki data kayıtları için index tutar. Bu formu kullanmasının nedeni, kayıtlara daha hızlı erişimin sağlanmasıdır.

5-Konum Bloğu: Btrieve, açılan herbir dosya ile bağlantılı konumlandırma bilgisi tutar. Bu bilgi, kayıt yöneticisi ile uygulama programı arasında geçen bir bellek bloğunda saklanır. Belleğin bu alanı KONUM BLOĞU olarak adlandırılır. Btrieve, dosya içinde gerçekleşen her işlemde konum bloğunu update eder. Buradaki bilgiler Btrieve tarafından, verilen bir erişim yolu (key) ile dosya üzerinde sıralı okuma yaparken kullanılır.

Konum bloğunda :

-Erişim yolu tanımlayıcısı
-Erişim yolu index pointer'leri
-Kayıt pointer'leri :

1-Önceki kayıt:Bir erişim yolundaki geçerli kayıtın önündeki kayıt
2-Geçerli kayıt:En son kullanılan kayıt

3-Sonraki kayıt:Bir erişim yolundaki geçerli kayıtı takip eden kayıt

1.2. RAIMA DATA MANAGER DATABASE MANAGEMENT SYSTEM (VERİ TABANI İŞLETME SİSTEMİ)

Raima Data Manager veri tabanı yönetim sistemi (DBMS), C dili uygulamalarının yüksek bir performansla yönetilmesini sağlayan önemli veri tabanı yönetim kapasitelerine sahip bir database'tir.

Tek bir sistemde Network (ağ) ve Relational (ilişkisel) model teknolojilerini birleştirilmiş olması, Raima'nın en önemli özelliklerinden biridir. Dataların karmaşıklığını gözönüne almaksızın bilgiye erişim ve organizasyonu en etkin biçimde yapmak istemesi, Raima Data Manager'ın bu iki teknolojiyi kullanmasındaki temel nedendir. Teknolojilerin birleştirilmesi, kullanıcıya müthiş derecede hız avantajı kazandırır. Ayrıca gereksiz data işlemlerini de minimize eder.

Data Management C'de yazılmıştır. Böylece uygulamaların VAX VMS, UNIX, QNX, Xenix, MS-DOS, OS/2 ve Microsoft Windows gibi işletim sistemleri altında çalışması kolaylaşmıştır.

Raima Data Manager içinde

- Sözlük
- Data dosyaları
- Key alanları (Index) bulunur.

Sözlük içerisinde, database'in tanımlama bileşenleri ve organizasyon bilgileri depolanır. Tüm kayıt tipleri bir data dosyasında, tüm anahtar alanları ise bir anahtar dosyasında saklanmaktadır.

1.2.1. RAIMA DATA MANAGER' IN ANA ÖZELLİKLERİ

1-LAN için çok kullanıcılı yapı desteği ve çok kullanıcılı bilgisayar çalıştırabilme özelliği.

2-Otomatik iyileştirme işlemi.

3-Network ve ilişkisel veri tabanı modellerine dayalı erişim metodları.

4-Çoklu veri tabanlarının artarak açılma ve kapanması.

5-Bir Database Manager veri tabanındaki bilgilerin daha kolay işlenmesi için etkileşimli ve toplu (batch) veri tabanı erişim utility'leri mevcuttur.

6-ASCII text dosyaları ve bir Database Manager veri tabanı arasında data geçişlerini yapabilmek için dosya transfer utility'leri mevcuttur.

7-Data saklamak için çoklu değerlerin kullanım özelliği.

8-Hızlı database erişimi için sanal bellek disk cache'ı.

9-Bütün database'i kontrol edebilmek için 100'ün üzerinde fonksiyona sahip bir çalışma kütüphanesi vardır.

10-Bir Data Manager veri tabanının yapısının ve içeriğinin tanımlanması için C programlama dilinden sonra, Database Definition Language (Veri tabanı Tanımlama Dili) adlı özel bir dile sahiptir.

11-Çoklu veri tabanı erişimi: Birden çok veri tabanı açılabilir ve uygulama programından aynı anda erişilebilir.

12-Çoklu veri tabanlarının artarak açılma ve kapanmaları.

1.2.2. RAIMA DATA MANAGER iLE İLGiLi TEMEL KAVRAMLAR

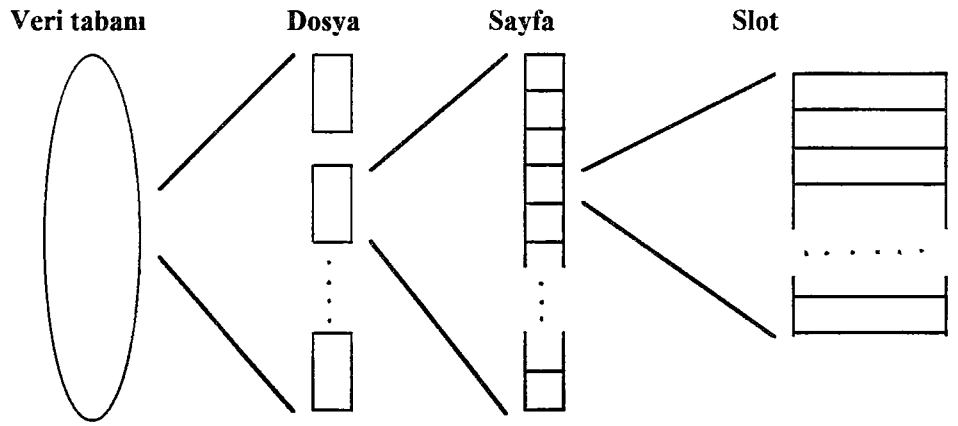
Bir database'in temel bilgi birimi **"FIELD"** (*alan*)'dır. Bir field (alan) ya da data field (data alan)'ı isim, tip, uzunluk gibi datanın temel özelliklerini taşır. Diğer veri tabanlarında field (alan) yerine *attribute* (özellik), *entity* (varlık) ve *column* (kolon-sütun) gibi değişik terimler kullanılmaktadır.

"RECORD" (*kayıt*) ise, birbiriyle bağlanıtılı alanlardan oluşan bir bütün olarak tanımlanabilir. Bazı veri tabanlarında kayıt terimi yerine *table* ya da *file* terimleri de kullanılmaktadır.

Bir kayıt üzerinde gerçekleşen tüm olaylar, bir işletim sistemi **DOSYA**sında saklanır. Dosyalar, bir veri tabanı yaratıldığında oluşturulan özel fiziksel saklama birimleridir. Bu nedenle bir veri tabanı için, birbirine bağlı dosyalardan oluşur denilebilir.

Raima Data Manager'da **DATA** ve **KEY** dosyası olmak üzere iki tip dosya bulunur. Kayıtlar sabit uzunlukludur. Eğer değişken uzunluklu kayıt kullanılırsa, en büyük kayıt uzunluğunda kayıtlar oluşturulur.

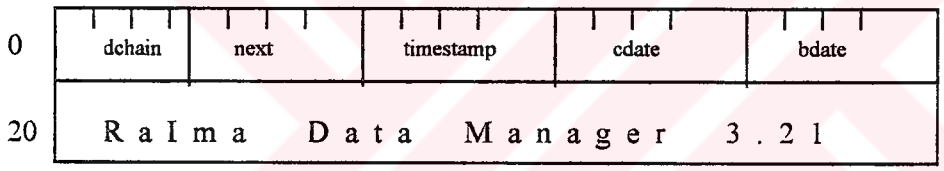
Aşağıdaki şekil, Data Manager tarafından kullanılan saklama birimlerini göstermektedir. Bir **VERİ TABANI** , bir ya da daha çok sayıda **DOSYA**dan oluşur. Bir dosya, bir ya da daha çok sayıda **SAYFA**dan ibarettir. Herbir sayfa ise eşit sayıda **SLOT**'a parçalanır ki bu parçalar bir data dosyasında **KAYIT**ları, bir anahtar dosyasında **ANAHTAR** değerlerini içerir.



Şekil-5 Bir Raima Data Manager Veritabanı'nın Birimleri

Tüm Data Manager dosyaları sabit uzunluktaki sayfaların bir sırası olarak görünür. Bir sayfanın uzunluğu dosyadan dosyaya değişkenlik gösterir. Ancak herbir dosya içinde tüm sayfalar aynı uzunluktadır. Tüm Database Manager veritabanı I/O' su, datanın sayfalarını tamamen okur ve yazar. Hatta kullanıcı seviyesindeki fonksiyonlar, kayıtlar ve anahtarlar üzerinde işlem yapar.

Sayfalar 0 (sıfir)'dan başlamak üzere numaralandırılır. Sıfırıncı sayfa dosyanın başıdır ve sadece dosya durum bilgilerini ihtiva eder. Bu sayfada saklanan özel bilgilere herhangi bir dosya erişemez. Aşağıdaki şekil sıfırıncı sayfayı, tablo ise bu sayfanın içeriğini göstermektedir.

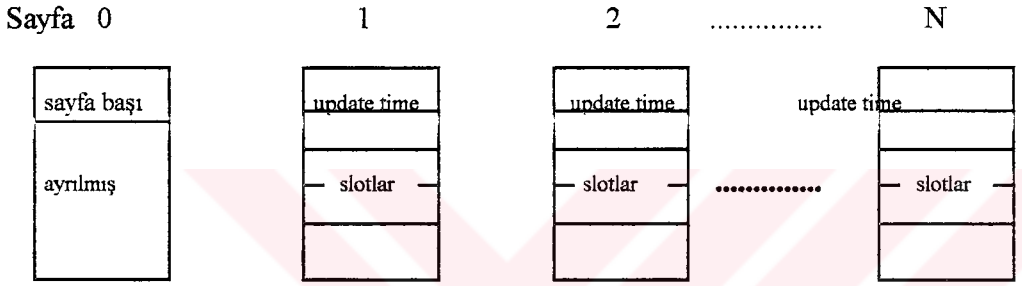


Şşekil-6 Sıfır no'lu nayfa

offset	Uzunluk	İsim	İçerik
0	4 Byte	dchain	silme zincirinin başı
4	4 Byte	next	dosyadaki bir sonraki geçerli boşluk
8	4 Byte	timestamp	sonraki timestamp yeri
12	4 Byte	cdate	Veri tabanının yaratılma tarihi
16	4 Byte	bdate	Veri tabanının son yedekleme tarih ve saati
20	21 Byte		Data Manager yazılımının geçerli versiyonu (ASCII)

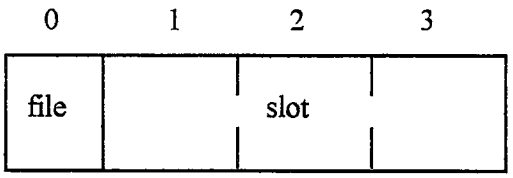
Şekil-7 Sıfır no'lu sayfanın içeriği

1 ya da üzeri sayı ile numaralandırılmış sayfaların tümünün ilk 4 Byte'ı bir Long Integer içerir. Bu değer, sayfanın update edildiği ve veri tabanına yazıldığı son tarih ve saati gösteren pul olarak servis verir. Herbir sayfanın geri kalan kısmı eşit uzunluktaki slotlara bölünür. Slotların büyüklüğü, dosyada saklanması gereken max. kayıt ya da anahtar büyüklüğüne göre belirlenir. Aşağıdaki şekil, genel bir Data Manager dosyasının katmanlarını göstermektedir.



Şekil-8 Bir Data Manager dosyasının katmanları

Raima Data Manager'ın data dosyasının ilk iki sıfır sayfa girişi, kayıt slot adreslerini içerir. Database'teki her kayıt slot'u, tekrarlanmaz şekilde isimlendirilir ve "Database Address" olarak bilinen 4 Byte'lık bir yapıda tutulur. Bir Database Adresi, bie dosya numarası ve bir slot numarasının biraraya gelmesinden oluşur. Aşağıdaki şekil bir Database Adresi'nin yapısını göstermektedir.



Şekil-9 Database address

Dosya numarası, 0 ile 255 arasında değer alan ve kayıtın saklandığı dosyayı gösteren işaretsiz bir sayıdır. Slot numarası ise, 1 ile 16.777.215 arasında değer alabilen (0. slot hariç) işaretsiz bir sayıdır. Bu sayı, o dosyada saklanabilecek maximum kayıt sayısını gösterir.

Bir kayıt database slot'una yazıldığında aşağıdaki yapıda bulunur:

offset	length	contents
0	2 Byte	kayıt tipi
2	4 Byte	bu kayıtın dbase adresi
6	4 Byte	kayıtın oluşma zamanı
10	4 Byte	kayıtın son update zamanı
(table)	variable	seçimlik anahtar bayrağı
(table)	variable	data değerleri

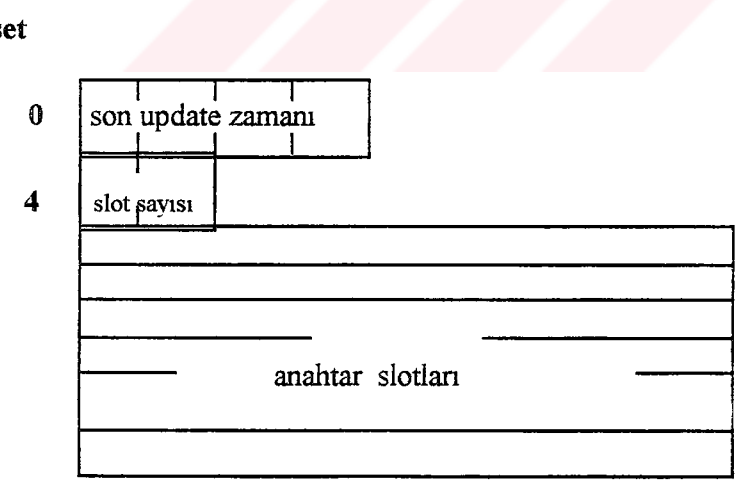
Dosyalarda saklanan bir kayıta erişim için kullanılan saha **"KEY"** (*anahtar*) olarak bilinir. Kayıt tanımları yapıldığı sırada, bu kayıtlara erişimi sağlayacak bir ya da daha fazla sayıda anahtar tanımına yer vermekte yarar vardır.

Bir kayıt için anahtar olarak kullanılan data sahası "key" özelliği taşır. db-Vista'da her alan anahtar olabilir. Böylece, özel bir kayıta erişim yolları sayısı çok geniş bir varyasyona sahip olur. Key alanı değerleri, data tipindeki doğal sırası ile key dosyasında saklanırlar. Eğer sadece unique keys e izin verliyorsa, alan bir **"UNIQUE KEY"** (tek, benzersiz) alanı olarak nitelenebilir. Bir anahtar alanının max. uzunluğu 246 byte'tır. Unique key olarak tanımlanan alanlar, halen key dosyasında bulunmayan bir

değer içermelidir. Eğer bir kayıt bir unique key alanındaki bir duplicate değer ile girilmek istenirse, durum kodu S_DUPLICATE gönderilir ve hata oluştuğundan kayıt girilmez.

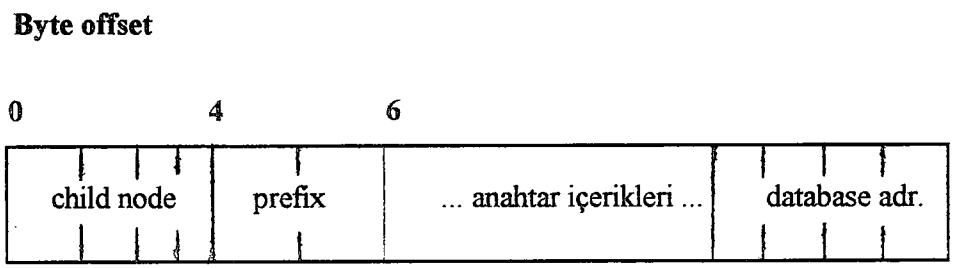
Bir anahtar dosyasında sıfır no'lu sayfadaki ilk iki giriş sayfa numaralarını içerir. B-tree yapısında tutulan anahtar dosyasında "node" adı verilen ve Data Manager anahtar dosyasının "sayfa" terimi ile eşanlamlı olan bir kelime kullanılmaktadır. Tüm Data Manager sayfalarında olduğu gibi, bir node 4 Byte'lık bir parça ile başlar. Bu alan son update zamanını tutar. Bu alanı takiben 2 Byte'lık bir alan gelir ki, bu alan node'da kullanılan geçerli slot'ların sayısını gösterir. Node'un geriye kalanı, sıralanmış anahtar slot'larının bir dizisidir. Bu arada "ORPHAN POINTER" terimi ile karşılaşıyoruz. Bu pointer'ın içerdiği değer, bu node'daki en büyük anahtardan daha büyük değere sahip anahtar node'larının sayısıdır. Bir node'da bulunabilecek maximum anahtar slotlarının sayısı, slot büyüklüğü ve node(sayfa) büyüklüğü ile belirlenir.

Aşağıdaki şekil, bir B-tree'de yeralan bir node'un genel olarak katmanlarını göstermektedir:



Şekil-10 Bir nodun katmanları

B-tree'de bir anahtar slot'unun yapısı ise şu şekilde gösterilebilir :



Şekil-11 Bir anahtar slot'unun yapısı

Child Pointer, bu slot'ta saklanan anahtar değerinden daha küçük anahtar değerlerini içeren node'un numarasıdır. Eğer child node yoksa, bu alan bir null node pointer (-1L) değerini içerir. Bir node'un child node'u yoksa, Leaf Node (Yaprak Node) olarak adlandırılır.

Prefix alanı, bir integer değerdir ve herbir anahtar tipi için unique (tekrarlanmaz)' dır. Bu alan, farklı türdeki anahtarlar aynı dosya içinde saklandığında, bunların sıralanması için kullanılır.

Anahtar değerleri ise data kayıtından bir ya da daha çok alanın bir kopyasıdır. Anahtar slot'unun son alanı, anahtarın data kayıtı ile ilişkili database adresidir.

Bu arada **Compound key** ve **Optional key** gibi iki anahtar türünün kullanıldığını görüyoruz. Burada bilinmesi gereken, optional keys kayıtlar oluşturulurken yaratılmaz. Daha çok real-time uygulamalarında kullanılır. Uygulama programı özel olarak istediğinde, d_keystore fonksiyonu yardımıyla yaratılır.

"INDEX" ise sadece anahtarları içeren bir dosyadır. Index, **"KEY FILE"**

(*anahtar dosyası*) terimi ile eş anlamlıdır.

"SCHEMA" (*şema*), bir database'in organizasyonu ve içeriğinin kavramsal tanımlamasıdır. Bir şema, alanları ve anahtarları ile birlikte tüm kayıt tiplerinin tanımını içerebilir: DBMS tarafından kullanılan şema formu **"DICTIONARY"** (*sözlük*) olarak adlandırılır.

Bir **"DATA MODELİ"** ya da **"DATABASE MODELİ"** , kayıtlar arası ilişkinin kavramsal bir betimlemesidir.İlişkisel database modeli, genel data dosyaları aracılığıyla kayıtlar arası ilişkileri düzenler ve bunların bakımını yapar. Diğer database modelleri, özellikle Network database modelinde kayıtlar arası ilişkiler direkt olarak

düzenlenir. Daha önce de belirttiğimiz gibi Raima Data Manager her iki modeli de destekler. Burada, Raima'nın desteklediği modeller hakkında bilgi vermek yararlı olacaktır :

AĞ VERİ TABANI YÖNETİM SİSTEMİ

Ağ veri tabanı veriyi, kayıt tiplerinin kümesi ve kayıt tipleri arasındaki iki yönlü ilişkiler şeklinde sunar. İkiden daha fazla kaydın bulunduğu ilişkiye doğrudan izin verilmez. Tüm ağ modelleri aynı kayıt tipleri arasında çoklu (1:n) ilişkiyi destekler. Basit ağ veri modelleri veri tabanı içinde (n:n) ilişkilerine doğrudan izin vermez. Karmaşık ağ veri modelleri bu tür ilişkilerin gösterilmesine izin verir. Ağ veri

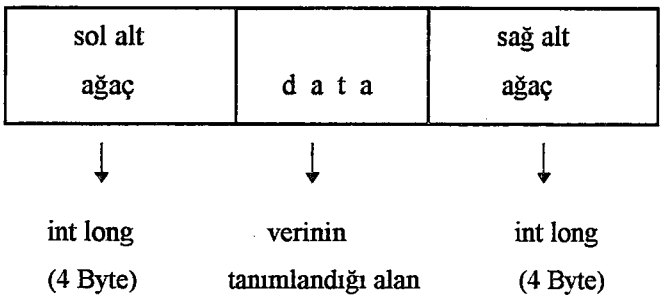
modeli karmaşık ve kullanılması zor bir modeldir. Ağ, hiyerarşinin genişletilmiş şekli olduğundan, ağ modelin anlamsal özellikleri hiyerarşik modelinkilere benzer.

İLİŞKİSEL VERİ TABANI YÖNETİM SİSTEMİ

İlişkisel veri tabanı hiyerarşik ve ağ modelinden farklıdır. İlişkisel model tablo küme'sinden oluşan bir veri tabanını göstermek için ilişki kavramını kullanır. İlişkideki her bir nesne değerler kümesi arasındaki ilişkiyi gösterir. İlişkisel veri tabanı yönetim sistemi ilk olarak mainframe bilgisayarlar ve iş veri işleme uygulamaları için tasarlanmıştır. Ayrıca ilişkisel sistemler çok sayıda kullanıcının kısa sorgulamalar kullandığı ortamlar için optimize edilmiştir. Bugünkü uygulamaların çoğu iş istasyonuna dayalıdır ve birkaç kullanıcıya hizmet eder. Bu uygulamalar veri tabanı tasarımını, multimedya sistemler ve bilgi tabanlarını içerirler. Karmaşık veri ve veri işlemleri içerirler. Örneğin, bir veri tabanı tasarımı bir çok parçadan oluşan nesneleri ve aynı nesnenin farklı versiyonlarını desteklemek zorundadır. Bir multimedya veri tabanı değişken uzunlukta text, grafik, görüntüler, ses ve video verisi içerebilir. Son olarak, bir veritabanı sistemi anlamsal olarak veri zenginliğine ihtiyaç duyar. Bu uygulamalarda ilişkisel DBMS'in kullanılması verinin kullanışsız bir şekilde ayrıştırılmasına ve zayıf bir performansa yol açar.

1.3. IPR (Internal Path Reduced) TREES (Kısaltılmış Dahili Yol)

Bilindiği üzere Binary Trees (ikili ağaçlar) kayıtları mantıksal olarak sıralamamıza imkan veren bir recursive veri yapısıdır. Bundan dolayı ikili ağaçlarla kayıt arama, yerleştirme ve silme işlemlerini hızlı bir şekilde yapabilmek mümkündür. İkili ağaçlarda herbir üye, solundaki ve sağındaki üyeyi gösteren bir link sahası içerir. Bu yapıyı basitçe göstermek gerekirse :



Şekil-12 B-tree'de kayıt deseni

İkili ağaçlarda her kayıt işleminden sonra dengeleme yapmak gerekir. Çünkü ikili ağaçların yapısı ve performansı, ağacın dengeli olması esasına dayanır. Ağaç içindeki herbir nod'un sol ve sağ alt ağaçlarının yükseklikleri aşağı yukarı birbirine eşitse bu ağaç dengelidir. Girilen veri çeşidi ne olursa olsun, ortalama arama sayısının en aza inmesi istendiği için, ağaç üzerinde daima binary search yapmak isteriz. Bir ağaçta bu işin yapılabilmesi için o ağacın dengeli olması gerekir. Bu nedenle, eğer herhangi bir kayıt eklendiğinde ağacın dengesi bozuluyor ise ağaç üzerinde dengeleme işlemi yapılır.

İkili ağaçlarda dengeleme olayı kayıtlar üzerindeki rotasyon olayına dayanır. Ağaçlarda yapılan rotasyon işleminin amacı, bir kayıta erişmek için gerekli probe sayısını azaltmaktır. Eğer ikili ağaç içindeki bir node'un ortalama derinliği minimuma indirilebilirse, probe sayısını da azaltmak mümkün olur. Çünkü bir node için gerekli probe sayısı doğrudan o node'un derinliğine bağlıdır. Ağaçtaki herbir node'un derinlikleri toplamı IPL (Internal Path Lenght) ne kadar küçük olursa, ağaçtaki kayıtlara daha kısa sürede erişmek mümkün olur. Bilindiği gibi IPR türü ikili ağaçlar, AVL türü ikili ağaçların aksine yükseklik dengeli değil ağırlık dengeli bir ağaçtır. Bundan dolayı, ağacın dengesiz olduğu durumlarda tekrar dengeyi sağlamak için rotasyonlar yapılır ve optimum performans sağlanır.

IPR ağaçlarda dengeleme işlemine karar vermeden önce herhangi bir node'un solundaki ve sağınadaki node sayısına bakılır. Bu ağaç tipinde herbir node üç gruba ayrılır :

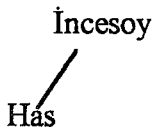
1-MR (more to rihgt): Node'un sağında bulunan kayıt sayısı solundaki kayıt sayısından fazladır.
2-ML (more to left): Node'un solundaki kayıt sayısı sağındaki kayıt sayısından fazladır.
3-Sağ ve soldaki kayıt sayıları eşit.

Ağacın MR olması durumunda tek veya çift rotasyon; ağacın ML olması durumunda ise sağa tek veya çift rotasyon durumu sözkonusu olabilir. Bu işlemlerin nasıl gerçekleştiği, tezin 3. bölümündeki kaynak program dökümlerinde görülebilir. Ancak olayın daha iyi anlaşılabilmesi için, rastgele seçilmiş soyisimleri IPR yöntemi ile dosyaya yazma örneği şu şekilde verilebilir :

n_a , n_b ve n_c , altağaçların nod sayıları olmak üzere, eğer $n_c > n_a$ ise ve ağaç ağırlığı sağda ise sola tek; eğer ağaç ağırlığı solda ise sağa tek rotasyon yapılır. Eğer $n_c < n_a$ ise $n_b > n_a$ kontrolü yapılır. Sonuç olumlu ise ve ağaç ağırlığı sağda ise sola çift; sağda ise sola çift rotasyon yapılır.

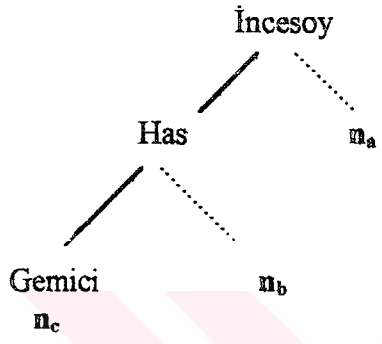
Soyisimler :İncesoy, Has, Gemici, Acar, German,Bolluca.

1. Önce İncesoy soyismi girilir. Bu soyisim ağacın kökü olur.
2. Sonra Has soyismi girilir. Kök ile Has kıyaslandığında, Has'ın İncesoy'dan küçük olduğu görülür ve İncesoy'un soluna nod olarak bağlanır. Has'ın adresi ise, İncesoy'un sol link alanına yazılır.

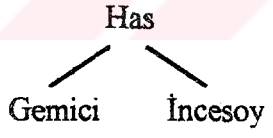


Bir ağaçta dengesizlik durumu oluşması için en az üç nod bulunması gerektiğinden

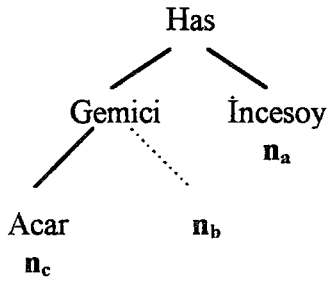
3. Gemici soyismi girilir. Gemici, önce İncesoy ile kıyaslanır. İncesoy'dan küçük olduğundan soluna bağlanması gerekmektedir. İncesoy'un sol linkinde Has'ın adresi vardır. Bu nedenle Gemici Has ile kıyaslanır. Has'tan da küçük olduğu için Has'ın soluna bağlanır ve Has'ın sol linkine Gemici'nin adresi yazılır. Ağaçta şimdi üç nod bulunmaktadır ve denge kontrolüne ihtiyaç vardır.



İncesoy kökü için n_a=0, n_b=0 ve n_c=1'dir. $n_c > n_a$ (1 > 0) olduğundan sağa tek rotasyon yapılır.

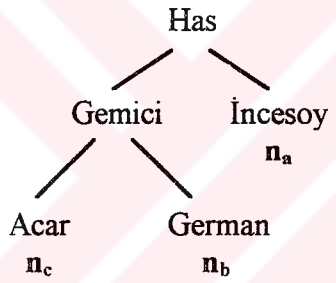


4. Acar soyismi girilir. Acar önce kök ile kıyaslanır. Kökten küçük olduğundan soluna bağlanacaktır. Has'ın sol linkinde Gemici'nin adresi bulunduğundan Acar bu nod ile kıyaslanır. Acar Gemici'den de küçük olduğu için bu nodun soluna bağlanır ve Gemici nodunun sol linkine Acar'ın adresi girilir.



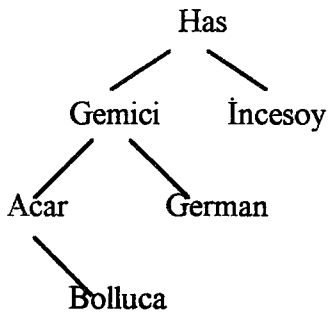
Has kökü için n_a=1, n_b=0 ve n_c=1'dir. n_c>n_a ve n_b>n_a şartları sağlanmadığından herhangi bir rotasyona gerek yoktur.

5. German girilir. Aynı algoritmaya göre German Gemici nodunun sağına bağlanır.



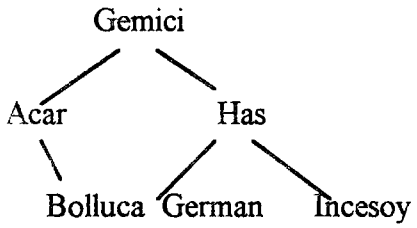
Has kökü için n_a=1, n_b=1 ve n_c=1 olduğundan herhangi bir rotasyona ihtiyaç yoktur.

6. Bolluca girilir. Algoritmaya göre Bolluca, Acar nodunun sağına bağlanır.



Gemici nodu için : n_a=1, n_b=1 ve n_c=0 olduğundan rotasyona gerek yoktur.

Has nodu için : n_a=1, n_b=1 ve n_c=2'dir. $n_c > n_a$ (2 > 1) olduğu için sağa tek rotasyon yapılır.



2. ANALİZ PROGRAMLARINA AİT ÖN BİLGİLER

2.1. ÇALIŞILAN BİLGİSAYAR HAKKINDA BİLGİ

Teze yönelik hazırlanan programların yazılıp derlenmesinde ve çalıştırılıp sonuçların elde edilmesinde kullanılan bilgisayar, 4 MB RAM belleğe, 120 MB HD'e sahip bir 486 DX-2 66'dır. Programların çalıştırılması esnasında bilgisayar Turbo'dan çıkartılmış ve 18 Mhz hıza düşürülmüştür.

2.2. PROGRAMLARDA KULLANILAN KAYIT DESENİ

Btrieve Record Manager, Raima Data Manager ve IPR Tree Metodu ile yazılan Sicil.C, Csicil.C ve Bsicil.C C programlarında kullanılan kayıt deseni aşağıdaki gibidir :

20 karakter 20 karakter (anahtar)

Bu kayıt deseninde yer alan Soyisim[20] parçası, yukarıda adı geçen programların hepsinde anahtar alan olarak kullanılmıştır. Anahtar alanın 0 ile 65.000 arasında değer alan integer tipi bir sayı olarak kullanıldığı Sicils.C, Bsicils.C ve Csicils.C programlarındaki kayıt deseni aşağıdaki gibidir :

NUMARA	İSİM
integer(anahtar)	20 karakter

2.3. BTRIEVE RECORD MANAGER'A AİT DOSYALAMA FONKSİYONLARI VE ÖRNEKLER

2.3.1. BTRIEVE RECORD MANAGER DOSYALAMA FONKSİYONLARI

Btrieve'de kayıtlar üzerinde yapılan her işlemin bir kodu vardır. Yani pek çok bilgisayar dilinde olduğu gibi, her işlemin İngilizce kelimelerden oluşan ayrı ayrı komutu yerine, her komutu temsil eden bir sayı mevcuttur.

Btrieve Record Manager'ın sahip olduğu işlem kodlarından BSİCİL.C programında kullanılanlar ve karşılıkları aşağıda verilmiştir :

0-OPEN :Var olan bir dosyayı açar.
1-CLOSE :Açık bir dosyayı kapatır.
2-INSERT :Açık olan dosyaya kayıt ekler.
4-DELETE :Aktif kaydı siler.
5-GET EQUAL :Bir kaydın eşitini getirir.
6-GET NEXT :Aktif kayıttan sonraki kaydı getirir.
12-GET FIRST :İlk kaydı getirir.
14-CREATE :Dosya yoksa yaratır, varsa üzerine tekrar açar.

2.3.2. FONKSİYONLARIN ÜRETTİĞİ STAT DEĞERLERİ

Btrieve, yukarıda örnekleri verilen her işlemin sonunda bir STAT (durum) değeri üretir. Bu değere göre program akışının yönü belirlenir. BSİCİL.C programında kullanılan STAT değerleri ve anlamları aşağıdaki gibi sıralanabilir :

09-End of File. (Dosya sonu)
12-File Not Found. (Dosya bulunamadı)
20-Record Manager Inactive. (Btrieve yüklenmemiş)
86-File Full. (Dosya dolu)

2.3.3. FONKSİYONLAR VE STAT DEĞERLERİ KULLANIMINA ÖRNEKLER

Yukarıda açıklanan işlem kodlarının ve STAT değerlerinin kullanımının daha iyi anlaşılması amacıyla, BSİCİL.C isimli programdan bazı bölümler alınarak, açıklamalarıyla birlikte örnek olarak verilmiştir :

1-) STAT = BTRV(**0**, pos_blk, &file_buf, &buf_len, file_name, 0);

if (STAT = = **20**) printf (" btrieve yüklenmemiş ..... % d " , stat);

if (STAT = = **12**) {

buf_len = size_of (file_buf);

STAT = BTRV(**14**, pos_blk, &file_buf, &buf_len, file_name,0);

if (STAT != **0**) printf (" dosya oluşturmada hata var %d " , stat);

STAT = BTRV(**0,** pos_blk, &file_buf, &buf_len,file_name,0);

if(STAT != **0**&&STAT != **86**) printf ("dosya açmada hata var %d ",stat); }

Açıklama: Yukarıdaki program parçasının ilk satırında 0 işlem kodlu OPEN fonksiyonunun kullanımı görülmektedir. Pozisyon bloğu, dosya buffer'ı, buffer uzunluğu ve dosya adı gibi terimlerin tanımlamaları ve değer atamaları, programın başında yapılmıştır. Bu tanımlara göre bu satırda Btrieve.Dat dosyasının açılışı gerçekleştirilmiştir. Program parçasının ikinci satırında, ilk satırın ürettiği STAT değerinin 20'ye eşit olup olmadığı kontrol edilir. Eğer eşitlik söz konusu ise, Btrieve Record Manager'ın yüklenmemiş olduğu uyarısı kullanıcıya iletilir. Üçüncü satırda ise, yine ilk satırın çalışması sonucu üretilen STAT değeri 12 sayısı ile karşılaştırılır. Çünkü, Btrieve yüklü olabilir ama açılmak istenen dosya henüz oluşturulmamış olabilir. Eğer böyle bir eşitlik durumu mevcutsa, dosya oluşturulmamıştır demektir. Bu durumda beşinci satırda görüldüğü üzere 14 işlem kodu kullanılarak daha önce adı tanımlanmış olan dosya yaratılır. Dosya yaratma işlemi sırasında herhangi bir hata oluşursa bunu kullanıcıya iletebilmek için, beşinci satırın ürettiği STAT değeri bu sefer 0 (sıfır) sayısı ile karşılaştırılır. Bilindiği gibi sıfır dışındaki her sayı bir hatanın oluştuğunu belirtmektedir. Ancak STAT değeri sıfıra eşit olduğunda hatasız bir yaratma işleminin gerçekleştiği anlaşılır. Program parçasının yedinci satırında hatasız olarak oluşturulmuş bir Btrieve'de açma işlemi yapılmaktadır. Takibeden satırda ise 0 (sıfır) ve 86 değerleri STAT ile kıyaslanarak, dosya açma işleminde harhangi bir hatanın olup olmadığı kontrol edilir.

2-)
```
kayit sayisi ( ) { int toplam_kayit = 0 ;
    STAT = BTRV(12, pos_blk, &sicbilgi, &buf_len, sicbilgi.soyadi,0);
    if ( STAT = = 9) { printf ( "dosyada kayit mevcut değil...."");
    return(0);        }
     do { toplam_kayit ++ ;
     STAT = BTRV( 6, pos_blk, &sicbilgi, buf_len, sicbilgi.soyadi,0);
         } while ( STAT != 9);
```

```
printf ( " toplam kayit sayisi = %d ", toplam_kayit);          }
```

Açıklama: Yukarıdaki program parçasının ikinci satırında 12 işlem kodu kullanılarak ilk kayıta konumlanma olayı gerçekleşmiştir. Daha sonra üçüncü satırda, ikinci satırın ürettiği STAT değeri 9 sayısı ile karşılaştırılarak, dosyada kayıt olup olmadığına bakılır. Aksi halde, bir kayıta erişim yapıldığından toplam kayıt sayısını tutan sayaç 1 arttırılır. Altıncı satırda ise dosyada bulunan bir sonraki kayıta erişim işlemi, 6 işlem kodu ile gerçekleştirilmiştir. Bir sonraki satırda ise, eğer dosya sonu değilse geri dönüp bir sonraki kayıta konumlanma işlemi için gereken kontrol yapılmaktadır.

3-)
```
do {  BTRV (4, pos_blk, &sicbilgi,&buf_len, sicbilgi.soyadi,0);
      STAT = BTRV(6, pos_blk, &sicbilgi, &buflen, sicbilgi.soyadi,0);
      } while ( STAT != 9);
```

Açıklama: Bu örnekte yapılan işlem kayıtların silinmesi işlemidir. Buna göre, ikinci satırdaki 4 işlem kodu, daha önce üzerine konumlanan kayıtı siler. Üçünci satırda ise bir sonraki kayıta konumlanılmıştır. Bir sonraki satırda 9 sayısı ile STAT değeri karşılaştırılır. Eğer eşitlik durumu olursa dosya sonuna gelindiği anlaşılır. Eşitsizlik durumunda ise bir sonraki kayıta erişilmek üzere geri dönülür.

4-)
```
t_ilk = clock( );
STAT = BTRV (2, pos_blk, &sicbilgi, &buf_len, sicbilgi.soyadi,0);
t_son = clock( );
```

Açıklama: 2 işlem kodu, açık olan dosyaya bir kayıt eklemek için kullanılır. Yukarıdaki üç satırı ile, bir kayıtın yazılması için geçen süre tutulmaktadır.

5-)
```
void silme ( ) {
```

```
        char soyisim [20] ;
        printf ( " silinecek soyadi giriniz : ");
        gets (sicbilgi.soyadi);
        clrscr( );
buf_len = sizeof (sicbilgi);
STAT = BTRV ( 5, pos_blk, &sicbilgi, &buf_len, sicbilgi.soyadi,0);
if ( STAT = = 0) {
  STAT = BTRV(4, pos_blk, &sicbilgi, &buf_len, sicbilgi.soyadi,0 );
  printf ( " istenen kayit silinmiştir..."\n);                    }
else  printf ( " silinecek kayit bulunamadi...%d \n        ", STAT);
printf ( " bir tuşa basiniz \n ");  getch( );
        }
```

Açıklama: Burada ise bir silme işlemi yapılmak isteniyor. Önce silinecek soyisimin ne olduğu sorulmuştur. Program parçasının yedinci satırında kullanılan 5 işlem kodu ile önce bu anahtar yardımıyla o kayıta konumlanmaya çalışılır. Eğer kayıt dosyada mevcutsa, 4 silme kodu yardımıyla silme işlemi gerçekleştirilir. Şayet kayıta ulaşılamıyorsa STAT değeri ile beraber mesaj olarak kullanıcıya iletilir.

6-)
```
void degistir ( ) {
    printf ( " kaydi degiştirilecek kişinin soyadini girin:");
    gets (sicbilgi.soyadi);
    STAT = BTRV(5, pos_blk, &sicbilgi, &buf_len, sicbilgi.soyadi,0);
      if ( STAT = = 0) {
            bilgi_al( );
            STAT = BTRV(3, &sicbilgi, &buf_len, sicbilgi.soyadi,0);
     if ( STAT ! = 0)   printf ( " update etmede hata var %d" , STAT);          }
     else { printf ( " aranan soyadi dosyada bulunamadi ");}
                  }
```

Açıklama: Bu örnekte ise yapılacak işlem update'tir. Öncelikle kayıta erişmek için anahtar alan olan soyadının girilmesi istenmiştir. Dördüncü satırda, 5 numaralı işlem kodu ile bu anahtarın temsil ettiği kayıtın eşitine erişilmektedir. Eğer işlem sırasında üretilen STAT sıfır ise kayıta ait yeni bilgiler klavyeden alınmaktadır. Değilse bu anahtara sahip kayıtın bulunamadığı mesajı verilir. Klavyeden alınan bilgiler daha sonra 3 numaralı işlem kodu yardımıyla dosyaya girilir. Tabi bu arada yine STAT'ın kontrolü yapılır. Sıfırdan farklı bir değer elde edilirse, update'in sağlıklı yapılmadığı kullanıcıya bildirilir.

7-) char database_close ( ) {
buf_len= sizeof(file_buf);
STAT = BTRV(1,pos_blk, &file_buf, &buf_len, file_name,0);
if (STAT ! = 0) printf ("dosya kapamada hata var .. %d ", STAT);
return(0); }

Açıklama: Bu örnekte ise, açık olan bir Btrieve dosyasının kapatılma işlemi gerçekleştirilmiştir. İşlem kodu olarak 1 kullanılmıştır. Dördüncü satırda yine işlemin doğruluğunu saptamak amacıyla STAT'ın sıfır ile karşılaştırması yaplmıştır.

2.4. RAIMA DATA MANAGER'A AİT DOSYALAMA FONKSİYONLARI VE ÖRNEKLER

2.4.1. RAIMA DATA MANAGER DOSYALAMA FONKSİYONLARI

d- open(database name,value)**:** Yaratılmış olan bir database'i açar. Database açmak için değişik amaçlara yönelik paramatreler mevcuttur. Bunlar :

x = exclusive access (özel erişim)

o = one user only (sadece bir kullanıcı için)

s = aynı anada birden çok database'in açılışı

d-close( ): Aktif database'i kapatır.

d-fillnew(REC, val, dbn): Yeni kayıt yaratılması ve değerlerinin yazılması.

d-recwrite(val,dbn): Geçerli kaydın değerlerini yazar.

d-delete(dbn): Geçerli kaydı siler.

d-recfrst (REC, dbn): REC kayıt tipinin ilk elemanına konumlanır.

d-recnext (dbn): Sonraki kayıtı bulma.

d-recread (val, dbn): Geçerli kayıtın içeriğini okuma.

d-crread (FLD, val, dbn): Geçerli kayıt alanından data okuma.

d-keyfind(key, val, dbn): *Be*nzer anahtarları bulma.

2.4.2. FONKSİYONLARIN ÜRETTİĞİ DURUM KODLARI VE ANLAMLARI

Raima Data Manager fonksiyonları işlem sonlarında bir takım hata kodları üretirler. Bu kodlardan 0 ile 9 arasında olanlar Fonksiyon Durum Kodları; -1 ile -50 arasında olanlar Kullanıcı Hata Kodları; -900 ile -924 arasında olanlar ise Sistem Hata Kodları olarak adlandırılır. SİCİL.C programında kullanılan kodlar ve anlamları aşağıdaki gibidir:

0 (S_OKAY):İşlem başarıyla tamamlandı. Herşey yolunda.

2 (S_NOTFOUND): Kayıt bulunamadı.

2.4.3. FONKSİYONLARIN VE STAT DEĞERLERİNİN KULLANIMINA ÖRNEKLER

1-) void database_open( ); {

```
extern int errno;
        if ( d_open ( " sicil " , " o ") != 0) {
            printf (" errno : %d \n ", errno);
            perror ("database açma hatası");    }   }
```

Açıklama: Yukarıdaki program parçasında SİCİL isimli database'in açılma işlemi örnek olarak verilmiştir. Görüldüğü gibi, üçüncü satırda açma fonksiyonunda, tek kullanıcı için database açıldığını belirten "o" parametresi kullanılmıştır. Eğer fonksiyon hatasız işlerse 0 (sıfir) durum kodunu üretir. Herhangi bir hatanın mevcudiyetinde ise hatayı kodu ile birlikte kullanıcıya iletir.

2-) void kayit_sayisi ( ) {

```
int toplam_kayit = 0;
for ( d_recfrst (kayit,CURR_DB); db_status= = s_okay; d_recnext (CURR_DB) )
    toplam_kayit + + ;
    printf ( " toplam kayıt sayısı = % d " , toplam_kayit);
}
```

Açıklama: Bu örnek ise, bir data dosyasına erişim işlemi ve bu işlem sırasında erişilen toplam kayıt sayısının hesaplandığı bir fonksiyondur. d_recfrst fonksiyonu, dosyadaki ilk kayıta erişimi sağlar. Eğer erişim olayı hatasız gerçekleşirse S_OKAY yani, 0 (sıfir) durum kodu üretilir. Hemen ardından dosyanın bir sonraki kayıtına erişmek için d_recnext fonksiyonu kullanılmıştır.

3-)
```
for ( i=0; i<10; i++ ) {
    zaman = 0;
    for ( j=0; j<400 ; j++) {
    fread ( sicbilgi.adi,20,1,dosyaad);
    fread (sicbilgi.soyadi.20,1,dosyasoy);
    t_ilk = clock( );
     d_fillnew (kayit, &sicbilgi,CURR_DB);
     t_son = clock( );
      zaman + =( t_son - t_ilk );
                        }
     }
```

Açıklama: Bu program parçası, ad ve soyad bilgilerinin ait oldukları dosyalardan 400'erli gruplar halinde okunup, database dosyasına kaydedilmesi işleminden alınmış bir parçadır. Bilindiği gibi d_fillnew fonksiyonu belirtilen database'e istenen formdaki kayıt (lar)ın girilmesini sağlar.

4-)
```
printf ( " aranacak soyadını giriniz :");
```

```
    gets ( soyisim );
    if ( d_keyfind ( SOYADI , soyisim, CURR_DB) = = S_NOTFOUND )  {
      printf ( " aranan soyadı bulunamadı ");                    }
    else  { d_recread (&sicbilgi, CURR_DB);
}
```

Açıklama: Yukarıdaki örnekte, Raima Data Manager'ın iki fonksiyonu kullanılmıştır. Çalışma konusu programlarda anahtar alan soyadı olduğundan, kayıt bulma işlemi için klavyeden soyisim değişkeni istenmiştir. Bu anahtara ait kayıtın varlığı d_keyfind fonksiyonu ile sağlanmıştır. Bu işlem sırasında, kayıtın bulunmama ihtimaline karşı stat değeri kontrol edilmiştir. Eğer kayıt bulunamazsa burada 2 stat koduna sahip S_NOTFOUND hatası ortaya çıkar. Şayet kayıt bulunursa, d_recread fonksiyonu ile bulunan kayıtın içeriği okunur.

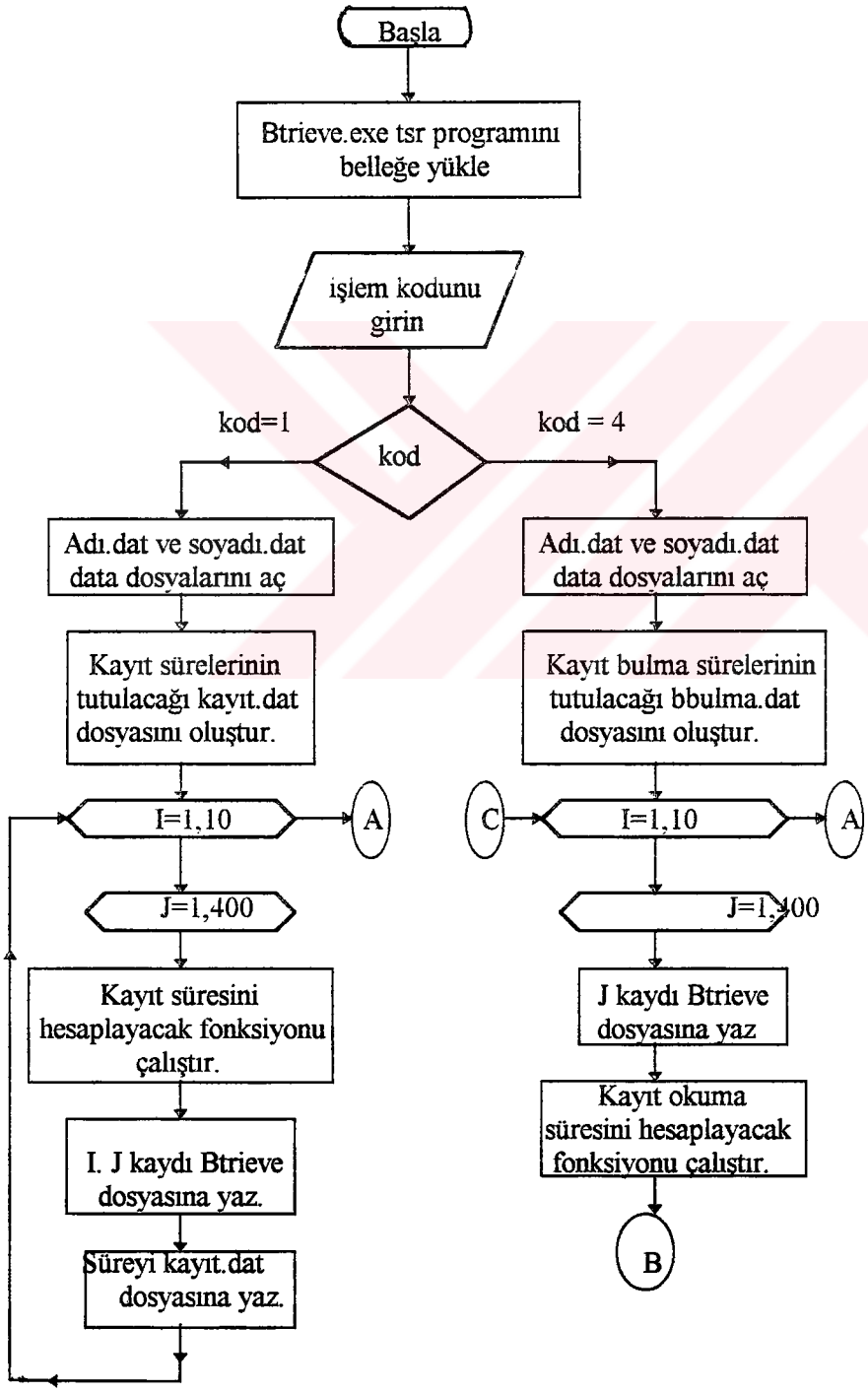
2.5. IPR YÖNTEMİ

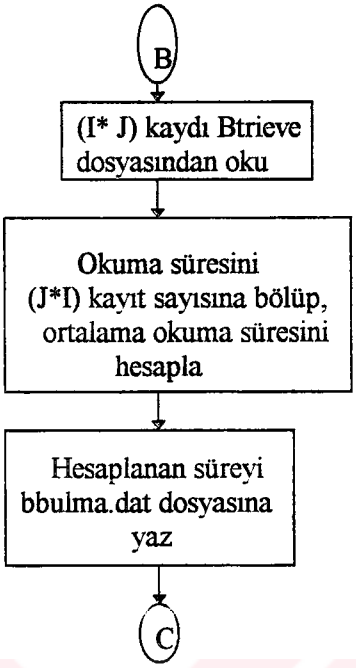
Bu yöntemin uygulandığı C programlarında, standart C dosyalama fonksiyonları kullanılmıştır. Kullanılan fonksiyonlar herkes tarafından bilindiğinden, bu bölümde ayrıca açıklanmayacaktır. Programların dökümü 3. bölümde yeralmaktadır.

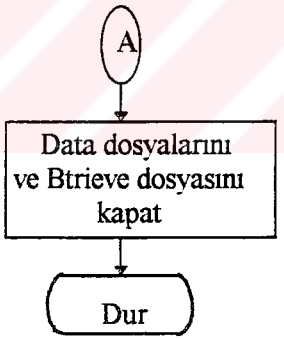
3. ANALİZ İÇİN YAZILAN PROGRAMLARIN AKIŞ ŞEMALARI VE DÖKÜMLERİ

3.1. PROGRAMLARIN AKIŞ ŞEMALARI

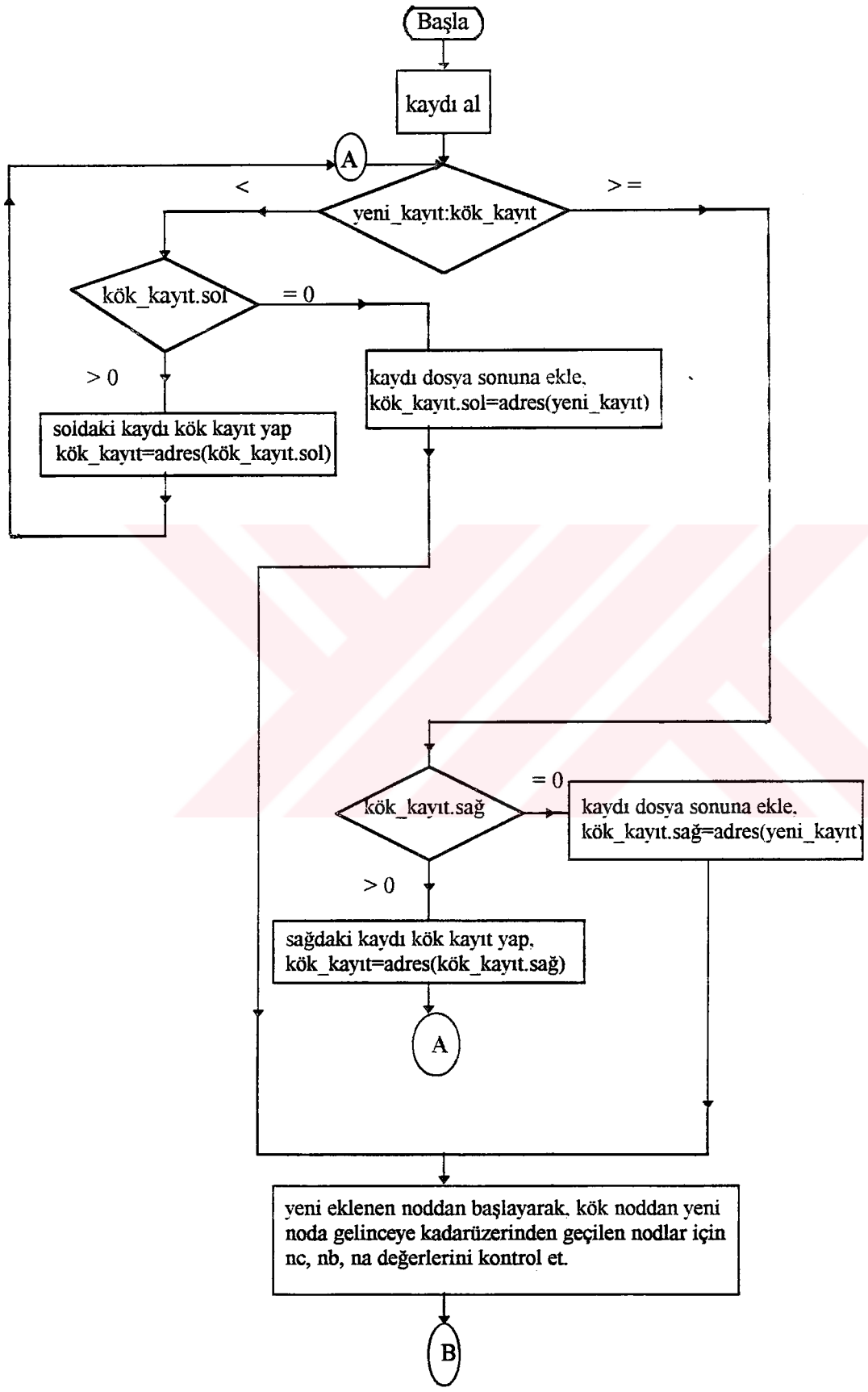
3.1.1. BSİCİL.C ve SİCİL.C PROGRAMLARININ AKIŞ ŞEMASI

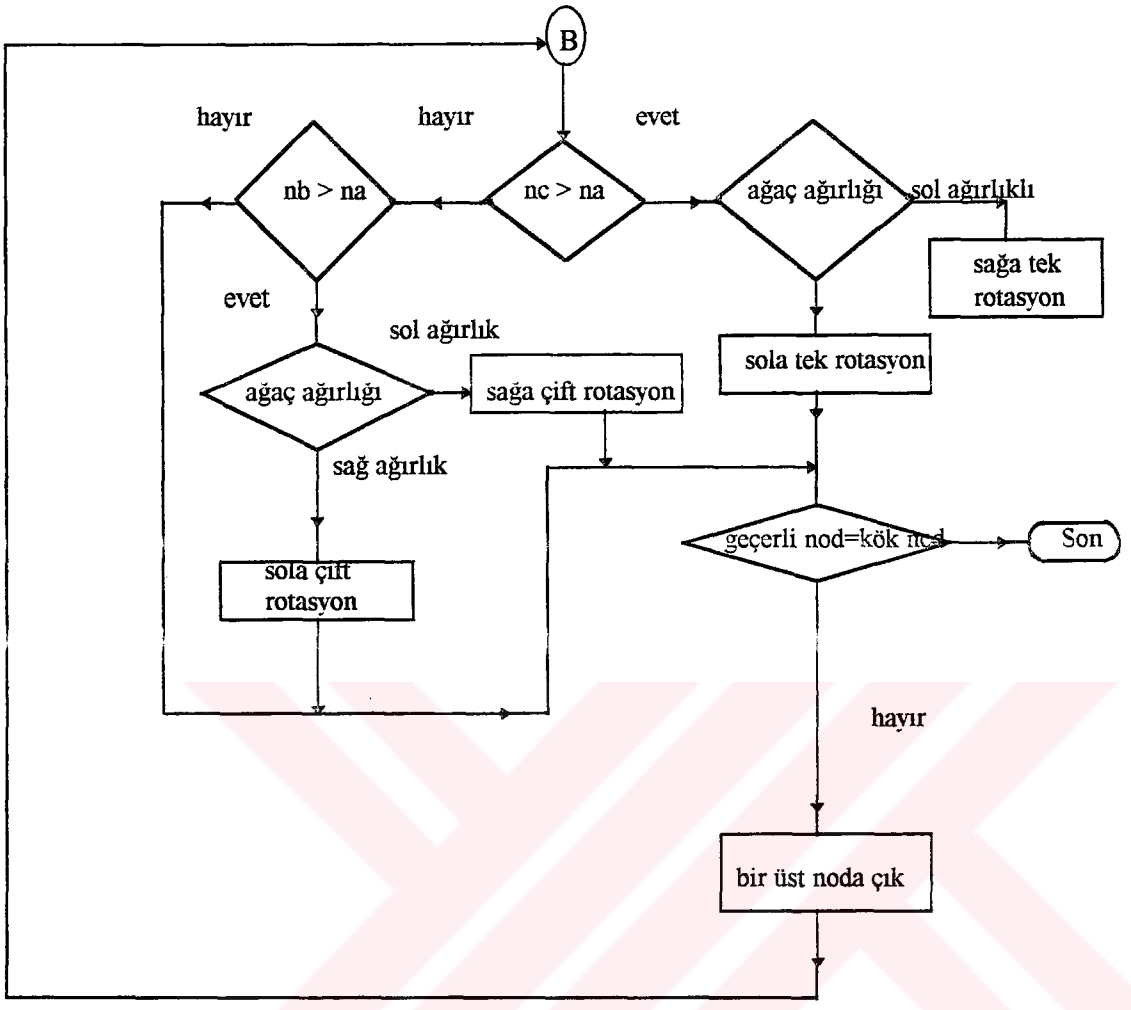



B
(I* J) kaydı Btrieve dosyasından oku
Okuma süresini (J*I) kayıt sayısına bölüp, ortalama okuma süresini hesapla
Hesaplanan süreyi bbulma.dat dosyasına yaz
C


A
Data dosyalarını ve Btrieve dosyasını kapat
Dur

3.1.2. CSİCİL.C PROGRAMININ AKIŞ ŞEMASI




B
hayır
hayır
evet
nb > na
nc > na
ağaç ağırlığı
sol ağırlıklı
sağa tek rotasyon
evet
sol ağırlık
ağaç ağırlığı
sağa çift rotasyon
sola tek rotasyon
sağ ağırlık
geçerli nod=kök ise
Son
sola çift rotasyon
hayır
bir üst noda çık

3.2. PROGRAMLARIN DÖKÜMLERİ

3.2.1. RANDOM.C

Btrieve, Raima Data Manager ve IPR trees yöntemlerini birbirleri ile kıyaslamak için 4000 adet rastgele kaydın bulunduğu dosya seçilmiştir. Bu kayıtları elle girmek yerine, kayıtları mümkün olduğu kadar birbirinden ayrı tutmak için bir program vasıtası ile random olarak girilmiştir. Program dökümü aşağıda belirtilmiştir.

Her üç yöntemde de sadece kayıt girişi ve kayıt okuma işlemleri kıyaslanmaktadır.

```
#include <stdlib.h>
#include <stdio.h>
#include <time.h>
char huge  dizi[5000][20];
char huge  dizi1[5000][20];
int esit(char huge *p1,char huge *p2)
{ register int i=0,j=0;
  while(p2[j]) j=j+1;
  while(p1[i]) i=i+1;
  if(i!=j)  return(0); // esit degil
  for(i=0;i<j;i++){
   if(p2[i])
   if(p1[i]!=p2[i]) { return(0); }  } // esit degil
 return(1);  //  esit dir
 }
void main(void)
{  FILE *dosya;
```

```
int i,j,es=0,sayac=0,k=0;
time_t t; char soyad[20];
// Asagiya cikti dosyasinin adini yaziniz
if ((dosya = fopen("soyadi.dat", "wt")) == NULL)
      {
fprintf(stderr, "Cikti dosyasi acilamadi\n");  }
srand((unsigned) time(&t));
        for(i=0; i<5000; i++) {
                for(j=0; j < 7+(random(14)); j++) {
                    dizi[i][j]= (65 + (rand() % 26));  }
                if (j==19)  dizi[i][19] =NULL; //string sonunu null'la
                else  dizi[i][j+1] = NULL;
          // for(j=0; j < 20; j++)
           //      printf("%c", dizi[i][j]);
           //      printf("\n");
                 }
      for(i=0; i<4000; i++) {  //  4000 tane unik isim sec
      for(j=i+1; j<5000; j++) {
       if(esit((char huge *) dizi[i],(char huge *) dizi[j]) == 1  ){
         printf("benzer string , elimine et, adet = %d\n",sayac); es =1 ; sayac++; break;
}
                }
        if (es == 0) {strcpy(dizi1[k++],dizi[i]);}
              es=0;
               }
      for(i=0; i<4000; i++) {  // isimleri ekrana yaz
              for(j=0; j<20; j++) {
              printf("%c",dizi1[i][j]);
               }
              printf("\n");}
```

```
        for(i=0; i<4000; i++) {
                for(j=0; j<20; j++) {
                    soyad[j] = dizi1[i][j];  }
        fwrite(soyad, 20, 1, dosya);  }
        fclose(dosya);
}
```

3.2.2. BSİCİL.C

Bu program, Btrieve kayıt yönetim sisteminin fonksiyonlarıyla C programlama dilinde yazılmıştır. Program, kayıt yazma ve kayıt okuma işlemlerini gerçekleştirmektedir. Bu arada, yapılan işlemlerin süreleri de tutulmaktadır.

```
#include<stdio.h>
#include<time.h>
#include"Turcbtrv.H"
#define TOTAL    4000
#define INTERVAL  400
FILE  *dosyaad , *dosyasoy;
FILE  *kaycikti , *bulcikti;
 struct KEY_SPEC
 {
  int KEY_POS;
  int KEY_LEN;
  int KEY_FLAG;
  char NOT_USE1[4];
  char KEY_TYPE;
  char RESERVE[5];
 };
```

```
struct FILE_SPEC
{
int REC_LEN;
 int PAGE_SIZ;
 int NDX_CNT;
 char NOT_USE2[4];
 int FILE_FLAG;
 char RESERVE2[2];
 int PRE_ALLOC;
 struct KEY_SPEC KEY_BUF[1]; /* ˜ndex Say□s□ */
};
struct kayit {
    char soyadi[20];
    char adi[20];
              } ;
int BUF_LEN;
struct FILE_SPEC  FILE_BUF;
char FILE_NAME[12]="btrieve.DAT";
char POS_BLK[128];
int STAT;
struct kayit sicbilgi = { "                ","                ",
"        ","          ","   ","       ","       ",
"        ","          ","   ","       ","       " };
clock_t t_ilk,t_son,zaman;
database_open() {
FILE_BUF.REC_LEN = sizeof(sicbilgi); /* Dosyada Kullan□lan Structur Uzunlu§u */
FILE_BUF.PAGE_SIZ = 1024; /* Bir Kerede Al□nan Kay□t Uzunlu§u */
FILE_BUF.FILE_FLAG = 0;
FILE_BUF.NDX_CNT = 1; /* Kullan□lan ˜ndex Say□s□ */
        /* Soyadina gore INDEX ISLEMLERI */
```

```
FILE_BUF.KEY_BUF[0].KEY_POS=1;   /* soyadinin Dosyadaki BaŸlang□‡ De§eri
*/ FILE_BUF.KEY_BUF[0].KEY_LEN=20;  /* ˜ndexlenecek Karekter Say□s□ */
FILE_BUF.KEY_BUF[0].KEY_FLAG=1|2|256; /* ˜ndex Cinsi */
FILE_BUF.KEY_BUF[0].KEY_TYPE=0;
BUF_LEN=sizeof(FILE_BUF);
STAT=BTRV(0,POS_BLK,&FILE_BUF,&BUF_LEN,FILE_NAME,0); //dosyayi ac
if (STAT==20) printf("Btrieve Yüklenmemiş , Btrieve'i yükleyiniz...%d",STAT);
if (STAT==12)  // dosya Bulunamadi  ise dosyayi olustur  ve sonra ac
{
  BUF_LEN=sizeof(FILE_BUF);
  STAT=BTRV(14,POS_BLK,&FILE_BUF,&BUF_LEN,FILE_NAME,0);
  if (STAT!=0)
          printf("Dosya Oluşturmada Hata Var %d",STAT);

  STAT=BTRV(0,POS_BLK,&FILE_BUF,&BUF_LEN,FILE_NAME,0);
  if (STAT!=0 && STAT!=86 ) printf("Dosya A‡mada Hata Var %d",STAT);
 }
 BUF_LEN=sizeof(sicbilgi);
 STAT=BTRV(12,POS_BLK,&sicbilgi,&BUF_LEN,sicbilgi.soyadi,0);
 return(0);
                          }
char database_close()  //closes btrieve files
{
 BUF_LEN=sizeof(FILE_BUF);
 STAT=BTRV(1,POS_BLK,&FILE_BUF,&BUF_LEN,FILE_NAME,0);
 if(STAT!=0) printf("Dosya Kapamada Hata Var! %d",STAT);
 return(0);
}
void data_dosyalarini_ac(){
  if ((dosyaad = fopen("adi.dat", "rt")) == NULL)
```

```
        { fprintf(stderr, "Cikti dosyasi acilamadi\n");  }
  if ((dosyasoy = fopen("soyadi.dat", "rt")) == NULL)
        { fprintf(stderr, "Cikti dosyasi acilamadi\n");  }
         }
void    data_dosyalarini_kapa(){
            fclose(dosyaad);
            fclose(dosyasoy);
}
void bilgi_yaz()  {   // kayit yapisini ekrana yazar
        printf("adi.....: %s\n",sicbilgi.adi);
        printf("soyadi....: %s\n",sicbilgi.soyadi);
                }
void bilgi_al() {     //  kayit yapisini ekrandan alir
        printf("\nadi.....:");  gets(sicbilgi.adi);
        printf("\nsoyadi.....:");  gets(sicbilgi.soyadi);

}
 kayit_sayisi() { int toplam_kayit=0;
        STAT=BTRV(12,POS_BLK,&sicbilgi,&BUF_LEN,sicbilgi.soyadi,0);  // ilk
kayda konumlan
        if (STAT== 9) {         printf("DOSYADA KAYIT MEVCUT
DEGIL...!!!!!");return(0);}
        do
         {
            toplam_kayit++;
        STAT=BTRV(6,POS_BLK,&sicbilgi,&BUF_LEN,sicbilgi.soyadi,0); // sonraki
kaydi oku
         }while(STAT!=9);
        printf("toplam kayit sayisi = %d ",toplam_kayit);
}
```

```
                /*****  Dosyadaki tum kayitlari sil  ********/
void sil_total_record(){
        kayit_sayisi();
        printf("\nSICIL DOSYASINDA BULUNAN TUM KAYITLAR
SILINIYOR!!!! ");
        STAT=BTRV(12,POS_BLK,&sicbilgi,&BUF_LEN,sicbilgi.soyadi,0);  // ilk
kayda konumlan
    if (STAT== 9) {  printf("DOSYADA KAYIT MEVCUT DEGIL...SILME
YOK!!!!!");return(0);}
        do
         {
        BTRV(4,POS_BLK,&sicbilgi,&BUF_LEN,sicbilgi.soyadi,0);  // Kayit siliniyor
        STAT=BTRV(6,POS_BLK,&sicbilgi,&BUF_LEN,sicbilgi.soyadi,0); // sonraki
kaydi oku
         }while(STAT!=9);  //  dosya sonu degilse geri don
}
                /******   KAYIT GIRME FONKSIYONU ***********/
void kayit() {
        int i,j;
        data_dosyalarini_ac();
  //    sil_total_record();  // dosyada kayit varsa sil
  if ((kaycikti = fopen("bkayit.dat", "wt")) == NULL)
        { fprintf(stderr, "Cikti dosyasi acilamadi\n");  }
//  initialize other fields here
        for(i=0; i<10; i++) {
        zaman=0;
        for(j=0; j< INTERVAL; j++) {
        fread(sicbilgi.adi, 20, 1, dosyaad);
        fread(sicbilgi.soyadi, 20, 1, dosyasoy);
        //bilgi_al();  // tekil kayit girmede kullanilmaktadir
```

```
        t_ilk = clock();
      STAT=BTRV(2,POS_BLK,&sicbilgi,&BUF_LEN,sicbilgi.soyadi,0);
        t_son = clock();
          if(STAT!=0)
                printf("Insert Etmede Hata Var!  %d",STAT);
        zaman += (t_son  - t_ilk) ;
        }
        fprintf(kaycikti,"\n- %d - ile - %d - arasindaki KAYIT icin gecen sure: %f\n", i *
INTERVAL,(i+1) *INTERVAL ,(zaman) / CLK_TCK);
        }
        data_dosyalarini_kapa();
        fclose(kaycikti);
}
void silme()  {
                char soyisim[20];
                printf("Silinecek soyadi giriniz..:");
                gets(sicbilgi.soyadi);
 clrscr();
 BUF_LEN=sizeof(sicbilgi);
 STAT=BTRV(5,POS_BLK,&sicbilgi,&BUF_LEN,sicbilgi.soyadi,0);
 if (STAT==0)
 {
  STAT=BTRV(4,POS_BLK,&sicbilgi,&BUF_LEN,sicbilgi.soyadi,0);
  printf("istenilen Kayıt Silinmiştir...\n");
 }
 else printf("Silinecek Kayıt Bulunamadı...%d\n",STAT);
 printf("Bir Tuşa Basınız\n");
 getch();
    }
void bulma(){
```

```
        int i,j;
        sil_total_record();  // dosyada kayit varsa sil
        printf("\nDosyadaki kayitlar silindi !!!");
        data_dosyalarini_ac();
  if ((bulcikti = fopen("bbulma.dat", "wt")) == NULL)
        { fprintf(stderr, "Cikti dosyasi acilamadi\n");  }
//  initialize other fields here
        for(i=0; i<10; i++) {
        zaman=0;
        fseek(dosyasoy,((long)((i)*((INTERVAL)*20L))), SEEK_SET);
        for(j=0; j< INTERVAL; j++) {
        fread(sicbilgi.soyadi, 20, 1, dosyasoy);
        STAT=BTRV(2,POS_BLK,&sicbilgi,&BUF_LEN,sicbilgi.soyadi,0);
        }
        fseek(dosyasoy, 0L, SEEK_SET);
        for(j=0; j< (i+1)*INTERVAL; j++) {
        fread(sicbilgi.soyadi, 20, 1, dosyasoy);
        t_ilk = clock();
        // Kayitlarin hepsinin bulundugu dogrulanmistir
  STAT=BTRV(5,POS_BLK,&sicbilgi,&BUF_LEN,sicbilgi.soyadi,0);
        t_son = clock();
          if(STAT!=0)
             {  printf("\nAranan soyadi dosyada bulunamadi!!! ");  }
        zaman += (t_son  - t_ilk) ;
                                          }
        fprintf(bulcikti,"\n-%d- ile -%d- arasindaki BULMA icin gecen sure:
%f\n",0,(i+1) *INTERVAL,((zaman) / CLK_TCK)/INTERVAL);
        }  // ilk for sonu
        data_dosyalarini_kapa();
        fclose(bulcikti);
```

```
        printf("\nBulma icin gerekli islemler tamamlandi  ");
          }
void degistir() {
        printf("\nKaydi degistirielcek kisinin soyadini giriniz..:");
        gets(sicbilgi.soyadi);
 STAT=BTRV(5,POS_BLK,&sicbilgi,&BUF_LEN,sicbilgi.soyadi,0);
  if (STAT==0) {
        bilgi_al();
        STAT=BTRV(3,POS_BLK,&sicbilgi,&BUF_LEN,sicbilgi.soyadi,0);
   if(STAT!=0)  printf("Update Etmede Hata Var! %d",STAT);
              }
   else {
        printf("\nAranan soyadi dosyada bulunamadi!!! "); }
}
main()
{   char komut[20];
        printf("Btrieve data ve index dosyasi siliniyor...!!!");
        system("del btrieve.dat");
    for( ; ; )   {
        printf("\n  ****  Asagidaki komutlardan birini seciniz  **** \n");
        printf("1-Sicil kaydi GIRME\n");
        printf("2-Sicil kaydi silme\n");
        printf("3-Sicil kaydi degistirme\n");
        printf("4-Sicil kaydi BULMA\n");
        printf("5-Tum Kayitlari silme\n");
        printf("6-Dosyadaki kayit sayisini ogrenme\n");
        printf("7-Cikis\n");
        printf("Bir komut giriniz( 1 ile 7 arasinda)\n");
        gets(komut);
        switch(komut[0]) {
```

```
        case '1':  database_open(); kayit(); database_close();  break;
        case '2':  database_open(); silme();    database_close(); break;
        case '3':  database_open(); degistir(); database_close(); break;
        case '4':  database_open(); bulma();    database_close();  break;
        case '5':  database_open(); sil_total_record(); database_close();  break;
        case '6':  database_open(); kayit_sayisi(); database_close();      break;
        case '7':  exit(0);
        default :  printf("Yanlis komut secildi!!!");
                }
}       }
```

3.2.3. SİCİL.C

DB-Vista kayıt yönetim sistemi fonksiyonlarıyla yazılmış, kayıt yazma ve okuma işlemlerinin yapıldığı C programı.

```
#include<stdio.h>
#include<time.h>
#include"c:\raima\include\vista.h"
#include"c:\nilgun\sicil.h"
#define TOTAL    4000
#define INTERVAL  400
FILE  *dosyaad , *dosyasoy;
FILE  *kaycikti , *bulcikti;
// dosya yapisinda struct tanimla ve gereksiz alanlarin ilk deger atamalarini
// yap
struct kayit sicbilgi = { "              ","              ",
"       ","          ","   ","       ","       ",
```

```
"       ","          ","   ","       ","       " };

clock_t t_ilk,t_son,zaman;
void   database_open() {
extern int errno;
                    if (d_open("sicil", "o") != 0) {
                      printf("errno: %d\n",errno);
                      perror("database acma hatasi ");
                      }
                          }
void data_dosyalarini_ac(){
  if ((dosyaad = fopen("adi.dat", "rt")) == NULL)
        { fprintf(stderr, "Cikti dosyasi acilamadi\n");  }
  if ((dosyasoy = fopen("soyadi.dat", "rt")) == NULL)
        { fprintf(stderr, "Cikti dosyasi acilamadi\n");  }
         }
void    data_dosyalarini_kapa(){
            fclose(dosyaad);
            fclose(dosyasoy);
}
void bilgi_yaz()  {   // kayit yapisini ekrana yazar
        printf("adi.....: %s\n",sicbilgi.adi);
        printf("soyadi....: %s\n",sicbilgi.soyadi);
                  }
void bilgi_al() {     //  kayit yapisini ekrandan alir
        printf("\nadi.....:");  gets(sicbilgi.adi);
        printf("\nsoyadi.....:");  gets(sicbilgi.soyadi);
}
void kayit_sayisi() { int toplam_kayit=0;
```

```
        for (d_recfrst(KAYIT,CURR_DB); db_status == S_OKAY ;
d_recnext(CURR_DB) )
            toplam_kayit++;
        printf("toplam kayit sayisi = %d ",toplam_kayit);
}
                /***** Dosyadaki tum kayitlari sil ********/
void sil_total_record(){
        kayit_sayisi();
        printf("\nSICIL DOSYASINDA BULUNAN TUM KAYITLAR
SILINIYOR!!!! ");
        for (d_recfrst(KAYIT,CURR_DB); db_status == S_OKAY ;
d_recnext(CURR_DB))
        d_delete(CURR_DB);
}
                /****** KAYIT GIRME FONKSIYONU ***********/
void kayit() {
        int i,j;
        data_dosyalarini_ac();
        sil_total_record();  // dosyada kayit varsa sil
  if ((kaycikti = fopen("kayit.dat", "wt")) == NULL)
        { fprintf(stderr, "Cikti dosyasi acilamadi\n");  }
//  initialize other fields here
        for(i=0; i<10; i++) {
        zaman=0;
        for(j=0; j< INTERVAL; j++) {
        fread(sicbilgi.adi, 20, 1, dosyaad);
        fread(sicbilgi.soyadi, 20, 1, dosyasoy);
        //bilgi_al();  // tekil kayit girmede kullanilmaktadir
```

```
        t_ilk = clock();
        d_fillnew(KAYIT,&sicbilgi,CURR_DB);
        t_son = clock();
        zaman += (t_son  - t_ilk) ;
        }

        fprintf(kaycikti,"\n- %d - ile - %d - arasindaki KAYIT icin gecen sure: %f\n", i *
INTERVAL,(i+1) *INTERVAL ,(zaman) / CLK_TCK);
        }
        data_dosyalarini_kapa();
        fclose(kaycikti);
}
void silme()  {
                char soyisim[20];
                printf("Silinecek soyadi giriniz..:");
                gets(soyisim);
if( d_keyfind(SOYADI,soyisim,CURR_DB) == S_NOTFOUND)  {
   printf("\nAranan soyadi dosyada bulunamadi!!! ");  }
else {  // aranan kayit bulundu ise
          d_recread(&sicbilgi,CURR_DB);
          bilgi_yaz();
          d_delete(CURR_DB);
  } }
void bulma(){
        int i,j;
        sil_total_record();  // dosyada kayit varsa sil
        printf("\nDosyadaki kayitlar silindi !!!");
        data_dosyalarini_ac();
  if ((bulcikti = fopen("bulma.dat", "wt")) == NULL)
```

```
        { fprintf(stderr, "Cikti dosyasi acilamadi\n");  }
//  initialize other fields here
        for(i=0; i<10; i++) {
        zaman=0;
        fseek(dosyasoy,((long)((i)*((INTERVAL)*20L))), SEEK_SET);
        for(j=0; j< INTERVAL; j++) {
        fread(sicbilgi.soyadi, 20, 1, dosyasoy);
        d_fillnew(KAYIT,&sicbilgi,CURR_DB);
        }
        fseek(dosyasoy, 0L, SEEK_SET);
        for(j=0; j< (i+1)*INTERVAL; j++) {
        fread(sicbilgi.soyadi, 20, 1, dosyasoy);
        t_ilk = clock();
        // Kayitlarin hepsinin bulundugu dogrulanmistir
if( d_keyfind(SOYADI,sicbilgi.soyadi,CURR_DB) == S_NOTFOUND)  {
   printf("\nAranan soyadi dosyada bulunamadi!!! ");  }
     //  printf("  %d  ",j);
        t_son = clock();
        zaman += (t_son  - t_ilk) ;
                                        }
        fprintf(bulcikti,"\n-%d- ile -%d- arasindaki BULMA icin gecen sure:
%f\n",0,(i+1) *INTERVAL,((zaman) / CLK_TCK)/INTERVAL);
        }  // ilk for sonu
        data_dosyalarini_kapa();
        fclose(bulcikti);
        printf("\nBulma icin gerekli islemler tamamlandi  ");
         // else {  // aranan kayit bulundu ise
         // d_recread(&sicbilgi,CURR_DB);
```

```
        // bilgi_yaz(); // sicil bilgilerini yaz
          }
                // }

void degistir() {
                char soyisim[20];
                printf("\nKaydi degistirielcek kisinin soyadini giriniz..:");
                gets(soyisim);
if( d_keyfind(SOYADI,soyisim,CURR_DB) == S_NOTFOUND)  {
   printf("\nAranan soyadi dosyada bulunamadi!!! ");  }
        else {  // aranan kayit bulundu ise
          bilgi_al();
          d_recwrite(&sicbilgi,CURR_DB);
           }
}
main( )
{   char komut[20];
                  database_open();
   kayit_sayisi();
   for( ; ; )  {
        printf("\n  ****  Asagidaki komutlardan birini seciniz  **** \n");
        printf("1-Sicil kaydi GIRME\n");
        printf("2-Sicil kaydi silme\n");
        printf("3-Sicil kaydi degistirme\n");
        printf("4-Sicil kaydi BULMA\n");
        printf("5-Tum Kayitlari silme\n");
        printf("6-Dosyadaki kayit sayisini ogrenme\n");
        printf("7-Cikis\n");
        printf("Bir komut giriniz( 1 ile 7 arasinda)\n");
```

```
gets(komut);
switch(komut[0]) {
  case '1':  kayit();   break;
  case '2':  silme();   break;
  case '3':  degistir(); break;
  case '4':  bulma();   break;
  case '5':  sil_total_record();   break;
  case '6':  kayit_sayisi();       break;
  case '7':  d_close(); exit(0);
  default  :  printf("Yanlis komut secildi!!!"); } }          }
```

3.2.4. CSİCİL.C

C programlama dilinin kendi fonksiyonlarıyla yazılmış kayıt yazma ve okuma programı. Programda kayıt yerleşimi IPR ikili ağaç yöntemine göre gerçekleştirilmiştir.

```
#include<stdlib.h>
#include<string.h>
#include<stdio.h>
#include<conio.h>
#include<time.h>
#include<io.h>
#define TOTAL    4000
#define INTERVAL  400
unsigned long  int  sola1=0,sola2=0,saga1=0,saga2=0;
struct tree {
   char soyad[20];
```

```
    char adi[20];
unsigned long int sayac;
    unsigned long int sol;
    unsigned long int sag;
} mem_kayit,temp,secondtemp,ayit,kayit = { "                ","                ",
"        "," "        "," "    "," "        "," "        ",
"        "," "        "," "    "," "        "," "        " };

clock_t  t_ilk,t_son,zaman;
unsigned long int mem_kok,b1,b2;        /* First  Node in  Tree        */
unsigned long int kayityeri;
/**************************          ******************************/
struct   file_bilgisi {
unsigned long int kok;
unsigned long int ilkyer;    /* First  node in delete chain */
unsigned long int kayit_say; /* Dosyada kayit varmi  yokmu  */
}mem_key,key;
unsigned long int balance(struct tree mem_kayit,unsigned long int mem_kok);
FILE *keyfile,*data,*dosyaad,*dosyasoy,*kaycikti,*bulcikti;
unsigned long filesize(FILE  *stream)   {
unsigned long curpos, length;
          curpos = ftell(stream);
          fseek(stream, 0L, SEEK_END);
          length = ftell(stream);
          fseek(stream, curpos, SEEK_SET);
          return length;
           }

void  data_file_ac()  {
   if((data  = fopen("IPLVERI.DAT","r+b")) == NULL)
```

```
        {    data=fopen("IPLVERI.DAT","w+b"); }
        }

void  key_file_ac(){
    if((keyfile = fopen("IPLKEY.DAT","r+b")) == NULL) {
        keyfile = fopen("IPLKEY.DAT","w+b");      }
    if (filesize(keyfile)==0L) {
        key.kok=0; mem_kok=0;
        key.ilkyer=0;
        key.kayit_say=0;
        fseek(keyfile,0L,SEEK_SET);
        fwrite(&key,sizeof(key),1,keyfile);
        }
    else if (filesize(keyfile)!=0L) {
        fseek(keyfile,0L,SEEK_SET);
        fread(&key,sizeof(key),1,keyfile); mem_kok=key.kok;  }
        }
/********************** STORE  TREE  FUNCTION
**********************/
unsigned long  int stree(unsigned long int mem_kok,unsigned long int  kayityeri,char
*soyad){
        struct  tree  mem_kayit;
        fseek(data,sizeof(kayit)*(mem_kok-1),SEEK_SET);
        fread(&mem_kayit,sizeof(kayit),1,data );
        mem_kayit.sayac++;
    if(strcmp(kayit.soyad,mem_kayit.soyad)<0) {
     if(mem_kayit.sol==0){
        mem_kayit.sol=kayityeri;
        kayit.sag=0;
        kayit.sol=0;
```

```
        kayit.sayac=1;
        fseek(data,sizeof(kayit)*(mem_kok-1),SEEK_SET);
        fwrite(&mem_kayit,sizeof(kayit),1,data );
        fseek(data ,sizeof(kayit)*(kayityeri-1),SEEK_SET);
        fwrite(&kayit,sizeof(kayit),1,data );/* return(kayit);*/
        return(mem_kok);
        }
    else  if(mem_kayit.sol!=0){
        mem_kayit.sol=stree(mem_kayit.sol,kayityeri,soyad);
        fseek(data ,sizeof(kayit)*(mem_kok-1),SEEK_SET);
        fwrite(&mem_kayit,sizeof(kayit),1,data );
        return(balance(mem_kayit,mem_kok));
        } }
  else if(strcmp(kayit.soyad,mem_kayit.soyad) >=0) {
   if(mem_kayit.sag==0){
        mem_kayit.sag=kayityeri;
        kayit.sol=0;
        kayit.sag=0;
        kayit.sayac=1;
        fseek(data,sizeof(kayit)*(mem_kok-1),SEEK_SET);
        fwrite(&mem_kayit,sizeof(kayit),1,data );
        fseek(data ,sizeof(kayit)*(kayityeri-1),SEEK_SET);
        fwrite(&kayit,sizeof(kayit),1,data );/*  return(kayit);*/
        return(mem_kok);}
   else if(mem_kayit.sag!=0){
        mem_kayit.sag=stree(mem_kayit.sag,kayityeri,soyad);
        fseek(data ,sizeof(kayit)*(mem_kok-1),SEEK_SET);/* Bir alt nod'da  herhangi
bir balance islemi dengeleme islemi olmus ise alt node degisebilir.*/
        fwrite(&mem_kayit,sizeof(kayit),1,data );
```

```
        return(balance(mem_kayit,mem_kok));
        }                    }
        return(mem_kok);
        }

unsigned long int balance(struct tree mem_kayit,unsigned long int mem_kok){
        unsigned long int
mem_sag,mem_sol,kay_sol,kay_sag,kayit_yeri,temp_yeri,donensol,donensag;
        struct tree kayit,temp;
if(mem_kayit.sol==0 && mem_kayit.sag==0){
        return(mem_kok); }
   else if(mem_kayit.sol!=0L){
        fseek(data,sizeof(kayit)*(mem_kayit.sol-1),SEEK_SET);
        fread(&kayit,sizeof(kayit),1,data);
        kayit_yeri=mem_kayit.sol;
        mem_sol=kayit.sayac;
        mem_sag=mem_kayit.sayac-kayit.sayac-1;
   if(mem_sol< mem_sag ) {
        fseek(data,sizeof(kayit)*(mem_kayit.sag-1),SEEK_SET);
        fread(&kayit,sizeof(kayit),1,data);
        kayit_yeri=mem_kayit.sag;
        mem_sag=kayit.sayac;
        mem_sol=mem_kayit.sayac-kayit.sayac-1;}
        }
   else if(mem_kayit.sol==0L) {
        fseek(data,sizeof(kayit)*(mem_kayit.sag-1),SEEK_SET);
        fread(&kayit,sizeof(kayit),1,data);
        kayit_yeri=mem_kayit.sag;
        mem_sag=mem_kayit.sayac-1;
        mem_sol=0;
```

```
        }
if (mem_sol==mem_sag) { return(mem_kok); /* Rotation Yok */ }
if (mem_sol>mem_sag) {   /* Sol  agirlik  ML tree */
 if(kayit.sol==0L && kayit.sag==0L){return(mem_kok);}
 if(kayit.sag!=0L){
        fseek(data,sizeof(kayit)*(kayit.sag-1),SEEK_SET);
        fread(&temp,sizeof(kayit),1,data);
        temp_yeri=kayit.sag;
        kay_sag=temp.sayac;
        kay_sol=kayit.sayac-temp.sayac-1;
        }
    else if(kayit.sag==0L){
        fseek(data,sizeof(kayit)*(kayit.sol-1),SEEK_SET);
        fread(&temp,sizeof(kayit),1,data);
        temp_yeri=kayit.sol;
        kay_sol=temp.sayac;
        kay_sag=kayit.sayac-temp.sayac-1;}
    if(kay_sol > mem_sag){  /* Tek  rotation  */
        sola1++;
        mem_kayit.sayac=mem_sag + kay_sag + 1 ;
        kayit.sayac=mem_kayit.sayac + kay_sol + 1 ;
        mem_kayit.sol=kayit.sag;
        kayit.sag=mem_kok;
        fseek(data,sizeof(kayit)*(mem_kok-1),SEEK_SET);
        fwrite(&mem_kayit,sizeof(kayit),1,data);
        fseek(data,sizeof(kayit)*(kayit_yeri-1),SEEK_SET);
        fwrite(&kayit,sizeof(kayit),1,data);
        donensag=balance(mem_kayit,mem_kok);
    if(donensag) {
        kayit.sag=donensag;
```

```
        fseek(data,sizeof(kayit)*(kayit_yeri-1),SEEK_SET);
        fwrite(&kayit,sizeof(kayit),1,data); }
        return(kayit_yeri);
        }
    else  if((kayit.sayac-kay_sol-1) > mem_sag) { /* iki  rotasyon */
        fseek(data,sizeof(kayit)*(kayit.sag-1),SEEK_SET);
        fread(&temp,sizeof(kayit),1,data);
        temp_yeri=kayit.sag;
    if (temp.sol==0 &&  temp.sag==0){
        b1=0;
        b2=0;
        }
    else  if (temp.sol!=0L) { /*   solunda  node  var */
        fseek(data,sizeof(kayit)*(temp.sol-1),SEEK_SET);
        fread(&secondtemp,sizeof(kayit),1,data);
        b1=secondtemp.sayac;
        b2=temp.sayac-secondtemp.sayac-1;
        }
    else  if(temp.sag!=0L) {  /* saginda  node  var */
        fseek(data,sizeof(kayit)*(temp.sag-1),SEEK_SET);
        fread(&secondtemp,sizeof(kayit),1,data);
        b2=secondtemp.sayac;
        b1=temp.sayac-secondtemp.sayac-1;
        }
        kayit.sayac=b1+kay_sol+1;
        mem_kayit.sayac=b2 + mem_sag+1;
        temp.sayac = kayit.sayac + mem_kayit.sayac+1;
        kayit.sag=temp.sol;
        temp.sol=kayit_yeri;
```

```
        mem_kayit.sol=temp.sag;
        temp.sag=mem_kok;
      sola2++;
        fseek(data,sizeof(kayit)*(temp_yeri-1),SEEK_SET);
        fwrite(&temp,sizeof(kayit),1,data);
        fseek(data,sizeof(kayit)*(kayit_yeri-1),SEEK_SET);
        fwrite(&kayit,sizeof(kayit),1,data);
        fseek(data,sizeof(kayit)*(mem_kok-1),SEEK_SET);
        fwrite(&mem_kayit,sizeof(kayit),1,data);
        donensag=balance(mem_kayit,mem_kok);
        donensol=balance(kayit,kayit_yeri);
    if(temp.sol!=donensol || temp.sag!=donensag) {
        temp.sol=donensol;
        temp.sag=donensag;
        fseek(data,sizeof(kayit)*(temp_yeri-1),SEEK_SET);
        fwrite(&temp,sizeof(kayit),1,data);        }
        return(temp_yeri);  }
     else  {        return(mem_kok);  /* Rotation yok    */  }
        }
else  if (mem_sol < mem_sag)  {      /* Sag agirlik MR  */
    if(kayit.sol==0L && kayit.sag==0L){return(mem_kok);}
    if(kayit.sol!=0L){
        fseek(data,sizeof(kayit)*(kayit.sol-1),SEEK_SET);
        fread(&temp,sizeof(kayit),1,data);
        temp_yeri=kayit.sol;
        kay_sol=temp.sayac;
        kay_sag=kayit.sayac-temp.sayac-1;}
    else if (kayit.sag!=0L){
        fseek(data,sizeof(kayit)*(kayit.sag-1),SEEK_SET);
```

```
        fread(&temp,sizeof(kayit),1,data);
        temp_yeri=kayit.sag;
        kay_sag=temp.sayac;
        kay_sol=kayit.sayac-temp.sayac-1;
        }
 if(kay_sag>mem_sol){  /* Tek  rotation */
        saga1++;
        mem_kayit.sayac=mem_sol + kay_sol + 1 ;
        kayit.sayac=mem_kayit.sayac + kay_sag + 1 ;
        mem_kayit.sag=kayit.sol;
        kayit.sol=mem_kok;
        fseek(data,sizeof(kayit)*(mem_kok-1),SEEK_SET);
        fwrite(&mem_kayit,sizeof(kayit),1,data);
        fseek(data,sizeof(kayit)*(kayit_yeri-1),SEEK_SET);
        fwrite(&kayit,sizeof(kayit),1,data);
        donensol=balance(mem_kayit,mem_kok);
if(donensol!=kayit.sol){
        kayit.sol=donensol;
        fseek(data,sizeof(kayit)*(kayit_yeri-1),SEEK_SET);
        fwrite(&kayit,sizeof(kayit),1,data); }
        return(kayit_yeri);
        }
else  if((kayit.sayac-kay_sag-1) > mem_sol) { /* iki rotasyon */

        fseek(data,sizeof(kayit)*(kayit.sol-1),SEEK_SET);
        fread(&temp,sizeof(kayit),1,data);
        temp_yeri=kayit.sol;
        if (temp.sol==0 &&  temp.sag==0){
        b1=0;
        b2=0;
```

```
        }
else if (temp.sol!=0L) { /*  solunda  node  var */
        fseek(data,sizeof(kayit)*(temp.sol-1),SEEK_SET);
        fread(&secondtemp,sizeof(kayit),1,data);
        b1=secondtemp.sayac;
        b2=temp.sayac-secondtemp.sayac-1;
        }
else if(temp.sag!=0L) {  /* saginda  node  var */
        fseek(data,sizeof(kayit)*(temp.sag-1),SEEK_SET);
        fread(&secondtemp,sizeof(kayit),1,data);
        b2=secondtemp.sayac;
        b1=temp.sayac-secondtemp.sayac-1;
        }
        kayit.sayac=b2 + kay_sag+1;
        mem_kayit.sayac=b1 + mem_sol+1;
        temp.sayac = kayit.sayac + mem_kayit.sayac+1;
        kayit.sol=temp.sag;
        temp.sag=kayit_yeri;
        mem_kayit.sag=temp.sol;
        temp.sol=mem_kok;
    saga2++;
        fseek(data,sizeof(kayit)*(temp_yeri-1),SEEK_SET);
        fwrite(&temp,sizeof(kayit),1,data);
        fseek(data,sizeof(kayit)*(kayit_yeri-1),SEEK_SET);
        fwrite(&kayit,sizeof(kayit),1,data);
        fseek(data,sizeof(kayit)*(mem_kok-1),SEEK_SET);
        fwrite(&mem_kayit,sizeof(kayit),1,data);
        donensol=balance(mem_kayit,mem_kok);
        donensag=balance(kayit,kayit_yeri);
 if(temp.sol!=donensol  ||  temp.sag!=donensag) {
```

```
        temp.sol=donensol;
        temp.sag=donensag;
        fseek(data,sizeof(kayit)*(temp_yeri-1),SEEK_SET);
        fwrite(&temp,sizeof(kayit),1,data); }
        return(temp_yeri);
        }
        else {      return(mem_kok); /* rotation yok */   } }
        }
                .
unsigned long int stree_ilk() {
        unsigned  long int  donen_kok;
    if  (key.kayit_say==0L)  {
        key.ilkyer=0; key.kayit_say=1;
        key.kok=1; kayit.sol=0; kayit.sag=0;
        kayit.sayac=1;
        fseek(data,0L,SEEK_SET);
        fwrite(&kayit,sizeof(kayit),1,data);
        fseek(keyfile,0L,SEEK_SET);
        fwrite(&key,sizeof(key),1,keyfile);
        //fclose(data );fclose(keyfile);
        return(1); }
    if (key.ilkyer==0L) {
        kayityeri=key.kayit_say+1;
        key.kayit_say+=1;
        fseek(keyfile,0L,SEEK_SET);
        fwrite(&key,sizeof(key),1,keyfile); }

    else if (key.ilkyer!=0L) {
        kayityeri=key.ilkyer;
        fseek(data,sizeof(kayit)*(key.ilkyer-1),SEEK_SET);
```

```
        fread(&mem_kayit,sizeof(kayit),1,data );
        key.ilkyer=mem_kayit.sag;
        //key.kayit_say+=1;
        fseek(keyfile,0L,SEEK_SET);
        fwrite(&key,sizeof(key),1,keyfile);
        }
        donen_kok=stree(mem_kok,kayityeri,kayit.soyad);
        if(key.kok!=donen_kok) {
        key.kok=donen_kok;
        fseek(keyfile,0L,SEEK_SET);
        fwrite(&key,sizeof(key),1,keyfile);}
        return(1);
        }

/*************** DELETE NODE FONKSİYONU ****************/

unsigned long  int dtree(unsigned long int mem_kok,char *soyad) {
        unsigned  long  int p,p2,eskisol,eskisag;
        struct  tree  mem_kayit,kayit; /*where wireless  tree  attached  to this  node  */
        fseek(data ,sizeof(mem_kayit)*(mem_kok-1),SEEK_SET);
        fread(&mem_kayit,sizeof(mem_kayit),1,data );
   if(strcmp(kayit.soyad,mem_kayit.soyad)==0)  {  /* delete  root */
        if(mem_kayit.sol==mem_kayit.sag) {  /* this  means  an  empty  tree */
        if(key.kayit_say==1L)  {
        key.kayit_say=0;
        key.ilkyer=0;//mem_kayit.sag=key.ilkyer;
        key.kok=0;
        }
        else  if(key.kayit_say!=1L)  {
        key.kayit_say-=1;  }
```

```
        mem_kayit.sag=key.ilkyer;
        key.ilkyer=mem_kok;
        fseek(data ,sizeof(mem_kayit)*(mem_kok-1),SEEK_SET);
        fwrite(&mem_kayit,sizeof(mem_kayit),1,data );
        fseek(keyfile,0L,SEEK_SET);
        fwrite(&key,sizeof(key),1,keyfile);
        return(0); }

 else if(mem_kayit.sol==0L){ /*  sağ ağaç varsa*/
        if(key.kok==mem_kok)
        key.kok=mem_kayit.sag;
        p=mem_kayit.sag;
        if(key.ilkyer==0)  mem_kayit.sag=0;
        else   mem_kayit.sag=key.ilkyer;
        key.ilkyer=mem_kok;
        key.kayit_say -=1;
        fseek(keyfile,0,SEEK_SET);
        fwrite(&key,sizeof(key),1,keyfile);
        fseek(data ,sizeof(mem_kayit)*(mem_kok-1),SEEK_SET);
        fwrite(&mem_kayit,sizeof(mem_kayit),1,data );
        //fclose(keyfile); fclose(data );
        return(p);  }
else  if(mem_kayit.sag==0L){
        if (key.kok==mem_kok)
        key.kok=mem_kayit.sol;
        p=mem_kayit.sol;
        if(key.ilkyer==0L)  mem_kayit.sag=0;
        else   mem_kayit.sag=key.ilkyer;
        key.ilkyer=mem_kok;
        key.kayit_say -=1;
```

```
        fseek(keyfile,0,SEEK_SET);
        fwrite(&key,sizeof(key),1,keyfile);
        fseek(data ,sizeof(mem_kayit)*(mem_kok-1),SEEK_SET);
        fwrite(&mem_kayit,sizeof(mem_kayit),1,data );
        return(p);
                        }

         /********  or both tree present     *********/
 else{  /********  Sag daki node kok olacak *********/
        if (key.kok==mem_kok)
        key.kok=mem_kayit.sag;
        p=mem_kayit.sag;
        p2=mem_kayit.sag;
        if(key.ilkyer==0) mem_kayit.sag=0;
        else  mem_kayit.sag=key.ilkyer;
        key.ilkyer=mem_kok;
        key.kayit_say -=1;
        fseek(data ,sizeof(mem_kayit)*(mem_kok-1),SEEK_SET);
        fwrite(&mem_kayit,sizeof(mem_kayit),1,data);
while(p!=0){
        fseek(data ,sizeof(kayit)*(p-1),SEEK_SET);
        fread(&kayit,sizeof(kayit),1,data );
        if (kayit.sol==0) break;
        else  p=kayit.sol;
        }
        kayit.sol=mem_kayit.sol;
        fseek(keyfile,0,SEEK_SET);
        fwrite(&key,sizeof(key),1,keyfile);
        fseek(data ,sizeof(mem_kayit)*(p-1),SEEK_SET);
        fwrite(&kayit,sizeof(mem_kayit),1,data);
```

```
        return(p2); }
        } /* ilk if in sonu */
else if (strcmp(kayit.soyad,mem_kayit.soyad)>0) { eskisag=mem_kayit.sag;
        mem_kayit.sag=dtree(mem_kayit.sag,kayit.soyad);
        if(eskisag!=mem_kayit.sag){
        fseek(data ,sizeof(mem_kayit)*(mem_kok-1),SEEK_SET);
        fwrite(&mem_kayit,sizeof(mem_kayit),1,data );}}
 else if(strcmp(kayit.soyad,mem_kayit.soyad)<0) { eskisol=mem_kayit.sol;
        mem_kayit.sol=dtree(mem_kayit.sol,kayit.soyad);
        if(eskisol!=mem_kayit.sol){
        fseek(data ,sizeof(mem_kayit)*(mem_kok-1),SEEK_SET);
        fwrite(&mem_kayit,sizeof(mem_kayit),1,data ); }}
        return(mem_kok);
        }

/************* SEARCH TREE FUNCTION **************/
unsigned long int search_tree(unsigned long int mem_kok,char *soyad) {

 if (!mem_kok) { return(0); }
        fseek(data ,sizeof(kayit)*(mem_kok-1),SEEK_SET);
        fread(&mem_kayit,sizeof(kayit),1,data );

 while (strcmp(mem_kayit.soyad,soyad)!=0) {
 if (strcmp(soyad,mem_kayit.soyad)<0) mem_kok=mem_kayit.sol ;
 else mem_kok=mem_kayit.sag;
 if (mem_kok==0) break;
        fseek(data ,sizeof(kayit)*(mem_kok-1),SEEK_SET);
        fread(&mem_kayit,sizeof(kayit),1,data );
        }
 if (strcmp(mem_kayit.soyad,soyad)== 0) { return(1); }
```

```
    else { return(0); }
                }
void data_dosyalarini_ac(){
  if ((dosyaad = fopen("adi.dat", "rt")) == NULL)
        { fprintf(stderr, "Cikti dosyasi acilamadi\n");  }
  if ((dosyasoy = fopen("soyadi.dat", "rt")) == NULL)
        { fprintf(stderr, "Cikti dosyasi acilamadi\n");  }
         }
void    data_dosyalarini_kapa(){
           fclose(dosyaad);
           fclose(dosyasoy);
}

void kaydet() {
        int i,j,STAT=0;
        data_dosyalarini_ac();
  if ((kaycikti = fopen("ckayit.dat", "wt")) == NULL)
        { fprintf(stderr, "Cikti dosyasi acilamadi\n");  }
        for(i=0; i<10; i++) {
        zaman=0;
        for(j=0; j< INTERVAL; j++) {
        fread(kayit.adi, 20, 1, dosyaad);
        fread(kayit.soyad, 20, 1, dosyasoy);
        key_file_ac(); data_file_ac();
        t_ilk = clock();
     STAT=stree_ilk();  // kaydi sakla
        t_son = clock();
        fclose(keyfile);
        fclose(data);
```

```
        if(STAT == 0)
            printf("Insert Etmede Hata Var!  %d",STAT);
        zaman += (t_son  - t_ilk) ;

        }
        fprintf(kaycikti,"\n- %d - ile - %d - arasindaki KAYIT icin gecen sure: %f\n", i *
INTERVAL,(i+1) *INTERVAL ,(zaman) / CLK_TCK);
        }
        data_dosyalarini_kapa();
        fclose(kaycikti);
}

void bulma(){
        int i,j,STAT=0;
        data_dosyalarini_ac();
  if ((bulcikti = fopen("cbulma.dat", "wt")) == NULL)
        { fprintf(stderr, "Cikti dosyasi acilamadi\n");  }
        for(i=0; i<10; i++) {
        zaman=0;
        fseek(dosyasoy,((long)((i)*((INTERVAL)*20L))), SEEK_SET);
        for(j=0; j< INTERVAL; j++) {
        key_file_ac(); data_file_ac();
        fread(kayit.soyad, 20, 1, dosyasoy);
        stree_ilk();
        fclose(keyfile);
        fclose(data);
        }
        fseek(dosyasoy, 0L, SEEK_SET);
```

```
        for(j=0; j< (i+1)*INTERVAL; j++) {
        fread(ayit.soyad, 20, 1, dosyasoy);
        key_file_ac(); data_file_ac();
        t_ilk = clock();
        STAT=search_tree(key.kok,ayit.soyad);
        t_son = clock();
          if(STAT==0)
             {  printf("\nAranan soyadi dosyada bulunamadi!!! ");  }
        zaman += (t_son  - t_ilk) ;
        fclose(keyfile);
        fclose(data);
                                        }
        fprintf(bulcikti,"\n-%d- ile -%d- arasindaki BULMA icin gecen sure:
%f\n",0,(i+1) *INTERVAL,((zaman) / CLK_TCK)/INTERVAL);

        }  // ilk for sonu
        data_dosyalarini_kapa();
        fclose(bulcikti);
        printf("\nBulma icin gerekli islemler tamamlandi  ");
          }
main()
{       char komut[20];
        key_file_ac(); data_file_ac();
   for( ; ; )  {
        printf("\n  ****  Asagidaki komutlardan birini seciniz  **** \n");
        printf("1-Sicil kaydi GIRME\n");
        printf("2-Sicil kaydi BULMA\n");
        printf("3-Cikis\n");
        printf("Bir komut giriniz( 1 ile 3 arasinda)\n");
        gets(komut);
```

```
        switch(komut[0]) {
                case '1': kaydet();  break;
                case '2': bulma();  break;
                case '3': fclose(keyfile);fclose(data); exit(0);
          default :  printf("Yanlis komut secildi!!!");

                }
}

}
```

3.2.5. RANDOMS.C

Bu program, Sicils.C, Bsicils.C ve Csicils.C programlarında anahtar alan olarak kullanılan Numara isimli ve değeri 0 ile 65.000 arasında değişen rastgele, 4000 adet inreger sayı üretmektedir.

```
#include <stdlib.h>
#include <stdio.h>
#include <time.h>
#define MAX1  16000
int huge  dizi1[4000];
int huge  dizi[16000];
void main(void)
{  FILE *dosya;
  int i,j,es=0,sayac=0,k=0;
  time_t t; int numara;

  // Asagiya cikti dosyasinin adini yaziniz
  if ((dosya = fopen("numara.dat", "wb")) == NULL)
        {
  fprintf(stderr, "Cikti dosyasi acilamadi\n");  }
                randomize();
                srand((unsigned) time(&t));
          for(i=0; i<8000; i++) {
                dizi[i]=(int) (rand() % 65535);
```

```
            }
//          for(i=0; i<4000; i++) {  // isimleri ekrana yaz
//          printf("%d",dizi[i]);
//          printf("\n");}

        for(i=0; i<MAX1-2; i++) {  //  4000 tane unik isim sec
        for(j=i+1; j<MAX1-1; j++) {
 if(dizi[i] == dizi[j]){
            printf("benzer deger , elimine et, adet = %d  , i,j degeri=%d  %d\n",sayac,i,j);
es =1 ; sayac++; break; }
                }
   if (es == 0) { dizi1[k++]=dizi[i]; }  //  benzeri yoksa elemani sec

                es=0;
                 if (k==4000) { break; }
                 }

//     for(i=0; i<4000; i++) {
  //printf("%d",dizi1[i]);
    //          printf("\n");}

        for(i=0; i<4000; i++) {
                        numara = dizi1[i];
                        printf("%d",dizi1[i]);
                        printf("\n");
        fwrite(&numara, sizeof(int), 1, dosya);  }

        fclose(dosya); }
```

3.2.6. SİCİLS.C

Bu program, anahtar alan olarak bir integer sayının kullanıldığı ve Btrieve kayıt fonksiyonları ile yazılmış bir C programıdır. Bu program sadece kayıt girişi ve kayıt okuma işlemlerini gerçekleştirmektedir.

```
#include<stdio.h>
#include<time.h>
#include"c:\btrv\Turcbtrv.H"
#define TOTAL     4000
#define INTERVAL  400
FILE  *dosyaad , *dosyanumara;
```

```
FILE  *kaycikti , *bulcikti;
 struct KEY_SPEC
 {
  int KEY_POS;
  int KEY_LEN;
  int KEY_FLAG;
  char NOT_USE1[4];
  char KEY_TYPE;
  char RESERVE[5];
 };

 struct FILE_SPEC
 {
  int     REC_LEN;
  int     PAGE_SIZ;
  int   NDX_CNT;
  char   NOT_USE2[4];
  int     FILE_FLAG;
  char   RESERVE2[2];



  int     PRE_ALLOC;
  struct KEY_SPEC KEY_BUF[1]; /* ˜ndex Say□s□ */
 };




 struct kayit {
    int  numara;
    char soyadi[20];
    char adi[20];
 } ;

          int      BUF_LEN;
          struct FILE_SPEC  FILE_BUF;
          char    FILE_NAME[12]="btrieve.dat";
          char    POS_BLK[128];
          int      STAT;

struct kayit sicbilgi = {0,"                  ","                  ",
"         ","             ","     ","         ","       ",
"         ","             ","     ","         ","       "};

clock_t t_ilk,t_son,zaman;
```

```
database_open() {
FILE_BUF.REC_LEN = sizeof(sicbilgi); /* Dosyada Kullan□lan Structur Uzunlu§u */
FILE_BUF.PAGE_SIZ = 1024; /* Bir Kerede Al□nan Kay□t Uzunlu§u */



FILE_BUF.FILE_FLAG = 0;
FILE_BUF.NDX_CNT = 1; /* Kullan□lan ˜ndex Say□s□ */

        /* numara ya gore INDEX ISLEMLERI */
FILE_BUF.KEY_BUF[0].KEY_POS=1;   /* numaranin Dosyadaki BaŸlang□‡ De§eri
*/
FILE_BUF.KEY_BUF[0].KEY_LEN= sizeof(int);  /* ˜ndexlenecek Karekter Say□s□
*/
FILE_BUF.KEY_BUF[0].KEY_FLAG=256; /* ˜ndex Cinsi */
FILE_BUF.KEY_BUF[0].KEY_TYPE=1;  //  integer icin key type numarasi

BUF_LEN=sizeof(FILE_BUF);
STAT=BTRV(0,POS_BLK,&FILE_BUF,&BUF_LEN,FILE_NAME,0); //dosyayi ac
if (STAT==20) printf("Btrieve Y□klenmemiŸ , Btrieve'i yukleyiniz...%d",STAT);
if (STAT==12)  // dosya Bulunamadi  ise dosyayi olustur  ve sonra ac
{
 BUF_LEN=sizeof(FILE_BUF);
 STAT=BTRV(14,POS_BLK,&FILE_BUF,&BUF_LEN,FILE_NAME,0);
 if (STAT!=0)
        printf("Dosya OluŸturmada Hata Var %d",STAT);

 STAT=BTRV(0,POS_BLK,&FILE_BUF,&BUF_LEN,FILE_NAME,0);
 if (STAT!=0 && STAT!=86 ) printf("Dosya A‡mada Hata Var %d",STAT);
}
BUF_LEN=sizeof(sicbilgi);
STAT=BTRV(12,POS_BLK,&sicbilgi,&BUF_LEN,&sicbilgi.numara,0);
return(0);
                        }

char database_close()  //closes btrieve files

{
BUF_LEN=sizeof(FILE_BUF);
STAT=BTRV(1,POS_BLK,&FILE_BUF,&BUF_LEN,FILE_NAME,0);
if(STAT!=0) printf("Dosya Kapamada Hata Var! %d",STAT);
return(0);
}


void data_dosyalarini_ac(){
```

```
   if ((dosyaad = fopen("adi.dat", "rt")) == NULL)
         { fprintf(stderr, "Adi veri dosyasi acilamadi\n");  }

   if ((dosyanumara = fopen("numara.dat", "rb")) == NULL)
         { fprintf(stderr, "Numara veri dosyasi acilamadi\n");  }
           }



void    data_dosyalarini_kapa(){
            fclose(dosyaad);
            fclose(dosyanumara);
}

void bilgi_yaz()  {   // kayit yapisini ekrana yazar
         printf("numara....: %d\n",sicbilgi.numara);
         printf("adi.......: %s\n",sicbilgi.adi);
         printf("soyadi....: %s\n",sicbilgi.soyadi);
                              }

void bilgi_al() {     //  kayit yapisini ekrandan alir
         printf("\numara.....:");  scanf("%d",&sicbilgi.numara);
         fflush(stdin);
         printf("\nadi.....:");  gets(sicbilgi.adi);
         printf("\nsoyadi.....:");  gets(sicbilgi.soyadi);
}

 kayit_sayisi() { int toplam_kayit=0;
         STAT=BTRV(12,POS_BLK,&sicbilgi,&BUF_LEN,&sicbilgi.numara,0);  // ilk
kayda konumlan
         if (STAT== 9) {         printf("DOSYADA KAYIT MEVCUT
DEGIL...!!!!!");return(0);}
         do
          {
             toplam_kayit++;
         STAT=BTRV(6,POS_BLK,&sicbilgi,&BUF_LEN,&sicbilgi.numara,0); //
sonraki kaydi oku
          }while(STAT!=9);

         printf("toplam kayit sayisi = %d ",toplam_kayit);
}

                  /*****  Dosyadaki tum kayitlari sil  ********/
void sil_total_record(){
         kayit_sayisi();
```

```
        printf("\nSICIL DOSYASINDA BULUNAN TUM KAYITLAR
SILINIYOR!!!! ");
        STAT=BTRV(12,POS_BLK,&sicbilgi,&BUF_LEN,&sicbilgi.numara,0);  // ilk
kayda konumlan
     if (STAT== 9) {  printf("DOSYADA KAYIT MEVCUT DEGIL...SILME
YOK!!!!!");return(0);}
        do
         {
        BTRV(4,POS_BLK,&sicbilgi,&BUF_LEN,&sicbilgi.numara,0);  // Kayit
siliniyor



        STAT=BTRV(6,POS_BLK,&sicbilgi,&BUF_LEN,&sicbilgi.numara,0); //
sonraki kaydi oku
         }while(STAT!=9);  //  dosya sonu degilse geri don

}
                /******  KAYIT GIRME FONKSIYONU ***********/
void kayit() {
        int i,j;
        data_dosyalarini_ac();


  //     sil_total_record();  // dosyada kayit varsa sil
  if ((kaycikti = fopen("bkayit.dat", "wt")) == NULL)
        { fprintf(stderr, "Cikti dosyasi acilamadi\n");  }
//  initialize other fields here
        for(i=0; i<10; i++) {
        zaman=0;
        for(j=0; j< INTERVAL; j++) {
        fread(sicbilgi.adi, 20, 1, dosyaad);
        fread(&sicbilgi.numara, sizeof(int), 1, dosyanumara);
        //bilgi_al();  // tekil kayit girmede kullanilmaktadir
        t_ilk = clock();

     STAT=BTRV(2,POS_BLK,&sicbilgi,&BUF_LEN,&sicbilgi.numara,0);
        t_son = clock();
          if(STAT!=0)
                printf("Insert Etmede Hata Var!  %d",STAT);

        zaman += (t_son  - t_ilk) ;
        }
        fprintf(kaycikti,"\n- %d - ile - %d - arasindaki KAYIT icin gecen sure: %f\n", i *
INTERVAL,(i+1) *INTERVAL ,((zaman) / CLK_TCK)/INTERVAL);

        }
        data_dosyalarini_kapa();
```

```
        fclose(kaycikti);
}

void silme() {
                printf("Silinecek numarayi giriniz..:");
                scanf("%d",&sicbilgi.numara);
              fflush(stdin);
 clrscr();
 BUF_LEN=sizeof(sicbilgi);
 STAT=BTRV(5,POS_BLK,&sicbilgi,&BUF_LEN,&sicbilgi.numara,0);
 if (STAT==0)
 {



  STAT=BTRV(4,POS_BLK,&sicbilgi,&BUF_LEN,&sicbilgi.numara,0);
  printf("~stenilen Kay□t Silinmi~tir...\n");
 }
 else printf("Silinecek Kay□t Bulunamad□...%d\n",STAT);
 printf("Bir Tu~a Bas□n□z\n");
 getch();
    }

void bulma(){
        int i,j;
        sil_total_record();  // dosyada kayit varsa sil
        printf("\nDosyadaki kayitlar silindi !!!");
        data_dosyalarini_ac();
  if ((bulcikti = fopen("bbulma.dat", "wt")) == NULL)
        { fprintf(stderr, "Cikti dosyasi acilamadi\n");  }
//  initialize other fields here
        for(i=0; i<10; i++) {
        zaman=0;
        fseek(dosyanumara,((long)((i)*((INTERVAL)*sizeof(int)))), SEEK_SET);

        for(j=0; j< INTERVAL; j++) {
        fread(&sicbilgi.numara, sizeof(int), 1, dosyanumara);
        STAT=BTRV(2,POS_BLK,&sicbilgi,&BUF_LEN,&sicbilgi.numara,0);
        }

        fseek(dosyanumara, 0L, SEEK_SET);
        for(j=0; j< (i+1)*INTERVAL; j++) {
        fread(&sicbilgi.numara, sizeof(int), 1, dosyanumara);
        t_ilk = clock();
        // Kayitlarin hepsinin bulundugu dogrulanmistir
  STAT=BTRV(5,POS_BLK,&sicbilgi,&BUF_LEN,&sicbilgi.numara,0);
        t_son = clock();
```

```
          if(STAT!=0)

              {  printf("\nAranan numara dosyada bulunamadi!!! ");  }

          zaman += (t_son  - t_ilk) ;
                                              }
          fprintf(bulcikti,"\n-%d- ile -%d- arasindaki BULMA icin gecen sure:
%f\n",0,(i+1) *INTERVAL,((zaman) / CLK_TCK)/(INTERVAL*(i+1)));


          }   // ilk for sonu
          data_dosyalarini_kapa();
          fclose(bulcikti);
          printf("\nBulma icin gerekli islemler tamamlandi  ");
            }



void degistir() {
          printf("\nKaydi degistirielcek kisinin numara giriniz..:");
          scanf("%d",&sicbilgi.numara);
 STAT=BTRV(5,POS_BLK,&sicbilgi,&BUF_LEN,&sicbilgi.numara,0);
  if (STAT==0) {
          bilgi_al();
          STAT=BTRV(3,POS_BLK,&sicbilgi,&BUF_LEN,&sicbilgi.numara,0);
   if(STAT!=0)  printf("Update Etmede Hata Var! %d",STAT);

              }
   else {
          printf("\nAranan numara dosyada bulunamadi!!! "); }

}

main()
{   char komut[20];


          printf("Btrieve data ve index dosyasi siliniyor...!!!");
          system("del btrieve.dat");
    for( ; ; )   {
          printf("\n  ****  Asagidaki komutlardan birini seciniz  **** \n");
          printf("1-Sicil kaydi GIRME\n");
          printf("2-Sicil kaydi silme\n");
          printf("3-Sicil kaydi degistirme\n");
          printf("4-Sicil kaydi BULMA\n");
          printf("5-Tum Kayitlari silme\n");
          printf("6-Dosyadaki kayit sayisini ogrenme\n");
```

```
        printf("7-Cikis\n");
        printf("Bir komut giriniz( 1 ile 7 arasinda)\n");
        gets(komut);
        switch(komut[0]) {
          case '1':  database_open(); kayit(); database_close();  break;
          case '2':  database_open(); silme();   database_close(); break;
          case '3':  database_open(); degistir(); database_close(); break;
          case '4':  database_open(); bulma();   database_close(); break;
          case '5':  database_open(); sil_total_record(); database_close();  break;
          case '6':  database_open(); kayit_sayisi(); database_close();     break;
          case '7':  exit(0);
          default :  printf("Yanlis komut secildi!!!");

                }
}           }
```

3.2.7. CSİCİLS.C

Bu program, anahtar alan olarak integer bir sayının kullanıldığı ve Vista kayıt fonksiyonları ile yazılmış bir C programıdır. Bu program sadece kayıt girişi ve kayıt okuma işlemlerini yapmaktadır.

```
#include<stdio.h>
#include<time.h>
#include"c:\vista\vista.h"
#include"c:\vista\sicil.h"
#define TOTAL     4000
#define INTERVAL  400
FILE  *dosyaad , *dosyanumara;
FILE  *kaycikti , *bulcikti;
// dosya yapisinda struct tanimla ve gereksiz alanlarin ilk deger atamalarini
// yap
struct kayit sicbilgi = { 0,"              ","              ",
"      ","      ","  ","  ","   ","   ",
"      ","      ","  ","  ","   ","   "};


clock_t t_ilk,t_son,zaman;
```

```
void   database_open() {
extern int errno;
                    if (d_open("sicil", "o") != 0) {
                      printf("errno: %d\n",errno);
                      perror("database acma hatasi ");
                      }
                           }

void data_dosyalarini_ac(){

  if ((dosyaad = fopen("adi.dat", "rt")) == NULL)
        { fprintf(stderr, "adi veri dosyasi acilamadi\n");  }

  if ((dosyanumara = fopen("numara.dat", "rb")) == NULL)

        { fprintf(stderr, "numara veri dosyasi acilamadi\n");  }

         }




void    data_dosyalarini_kapa(){
           fclose(dosyaad);
           fclose(dosyanumara);
}

void bilgi_yaz()  {   // kayit yapisini ekrana yazar
        printf("adi.....: %s\n",sicbilgi.adi);
        printf("soyadi....: %s\n",sicbilgi.soyadi);


                    }

void bilgi_al() {     //  kayit yapisini ekrandan alir
        printf("\nadi.....:");  gets(sicbilgi.adi);
        printf("\nsoyadi.....:");  gets(sicbilgi.soyadi);

}

void kayit_sayisi() { int toplam_kayit=0;
        for (d_recfrst(KAYIT,CURR_DB); db_status == S_OKAY ;
d_recnext(CURR_DB) )
           toplam_kayit++;
        printf("toplam kayit sayisi = %d ",toplam_kayit);
```

```
}


                /***** Dosyadaki tum kayitlari sil ********/
void sil_total_record(){
        kayit_sayisi();
        printf("\nSICIL DOSYASINDA BULUNAN TUM KAYITLAR
SILINIYOR!!!! ");
        for (d_recfrst(KAYIT,CURR_DB); db_status == S_OKAY ;
d_recnext(CURR_DB))
        d_delete(CURR_DB);
}

                /******  KAYIT GIRME FONKSIYONU ***********/
void kayit() {
        int i,j;
        data_dosyalarini_ac();
        sil_total_record();  // dosyada kayit varsa sil
  if ((kaycikti = fopen("kayit.dat", "wt")) == NULL)
        { fprintf(stderr, "Cikti dosyasi acilamadi\n");  }

//  initialize other fields here
        for(i=0; i<10; i++) {
        zaman=0;
        for(j=0; j< INTERVAL; j++) {
        fread(sicbilgi.adi, 20, 1, dosyaad);
        fread(&sicbilgi.numara, sizeof(int), 1, dosyanumara);
        //bilgi_al();  // tekil kayit girmede kullanilmaktadir
        t_ilk = clock();
        d_fillnew(KAYIT,&sicbilgi,CURR_DB);
        t_son = clock();
        zaman += (t_son  - t_ilk) ;
        }
        fprintf(kaycikti,"\n- %d - ile - %d - arasindaki KAYIT icin gecen sure: %f\n", i *
INTERVAL,(i+1) *INTERVAL ,((zaman) / CLK_TCK)/INTERVAL);
        }
        data_dosyalarini_kapa();
        fclose(kaycikti);
}

void silme()  {
                int sayi;
                printf("Silinecek soyadi giriniz..:");
                scanf("%d",&sayi);
                printf("Inputed : %d",sayi);
```

```
if( d_keyfind(NUMARA,&sayi,CURR_DB) == S_NOTFOUND)  {
   printf("\nAranan soyadi dosyada bulunamadi!!! ");  }
else {  // aranan kayit bulundu ise
          d_recread(&sicbilgi,CURR_DB);
          bilgi_yaz();
          d_delete(CURR_DB);
   } }

void bulma(){
        int i,j;
        sil_total_record();  // dosyada kayit varsa sil
        printf("\nDosyadaki kayitlar silindi !!!");
        data_dosyalarini_ac();
  if ((bulcikti = fopen("bulma.dat", "wt")) == NULL)
        { fprintf(stderr, "Cikti dosyasi acilamadi\n");  }
//  initialize other fields here


        for(i=0; i<10; i++) {
        zaman=0;
        fseek(dosyanumara,((long)((i)*((INTERVAL)*sizeof(int)))), SEEK_SET);



        for(j=0; j< INTERVAL; j++) {

        fread(&sicbilgi.numara, sizeof(int), 1, dosyanumara);
        d_fillnew(KAYIT,&sicbilgi,CURR_DB);
        }

        fseek(dosyanumara, 0L, SEEK_SET);
        for(j=0; j< (i+1)*INTERVAL; j++) {
        fread(&sicbilgi.numara, sizeof(int), 1, dosyanumara);
        t_ilk = clock();
        // Kayitlarin hepsinin bulundugu dogrulanmistir
if( d_keyfind(NUMARA,&sicbilgi.numara,CURR_DB) == S_NOTFOUND)  {
   printf("\nAranan soyadi dosyada bulunamadi!!! ");  }

     //  printf("  %d  ",j);
        t_son = clock();
        zaman += (t_son  - t_ilk) ;
                                        }
        fprintf(bulcikti,"\n-%d- ile -%d- arasindaki BULMA icin gecen sure:
%f\n",0,(i+1) *INTERVAL,((zaman) / CLK_TCK)/(INTERVAL*(i+1)));


        }  // ilk for sonu
```

```
        data_dosyalarini_kapa();
        fclose(bulcikti);
        printf("\nBulma icin gerekli islemler tamamlandi  ");

         // else {  // aranan kayit bulundu ise
         // d_recread(&sicbilgi,CURR_DB);
         // bilgi_yaz(); // sicil bilgilerini yaz
           }

                 //  }

void degistir() {
                int sayi;
                printf("\nKaydi degistirielcek kisinin soyadini giriniz..:");
                scanf("%d",&sayi);
                printf("Inputed : %d",sayi);

if( d_keyfind(NUMARA,&sayi,CURR_DB) == S_NOTFOUND)  {
   printf("\nAranan soyadi dosyada bulunamadi!!! ");  }
        else {  // aranan kayit bulundu ise
         bilgi_al();
         d_recwrite(&sicbilgi,CURR_DB);
          }




}

main()
{   char komut[20];
                  database_open();
   kayit_sayisi();
   for( ; ; )  {
        printf("\n  ****  Asagidaki komutlardan birini seciniz  **** \n");
        printf("1-Sicil kaydi GIRME\n");
        printf("2-Sicil kaydi silme\n");
        printf("3-Sicil kaydi degistirme\n");
        printf("4-Sicil kaydi BULMA\n");
        printf("5-Tum Kayitlari silme\n");
        printf("6-Dosyadaki kayit sayisini ogrenme\n");
        printf("7-Cikis\n");
        printf("Bir komut giriniz( 1 ile 7 arasinda)\n");
        gets(komut);
        switch(komut[0]) {
         case '1':  kayit();   break;
         case '2':  silme();   break;
```

```
            case '3':  degistir(); break;
            case '4':  bulma();   break;
            case '5':  sil_total_record();   break;
            case '6':  kayit_sayisi();       break;
            case '7':  d_close(); exit(0);
            default  :  printf("Yanlis komut secildi!!!");

                    }
}           }
```

3.2.8. SİCİLS.C

Bu program, kayıt yerleşiminin IPR metodu ile elle yapıldığı ve anahtar alan olarak integer bir sayının tanımlandığı C programıdır. Bu programda sadece kayıt girişi ve kayıt okuma işlemleri yapılmaktadır.

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



#include<conio.h>
#include<time.h>
#include<io.h>
#define TOTAL    4000
#define INTERVAL  400
int ii=0;
unsigned  long   int   sola1=0,sola2=0,saga1=0,saga2=0;

struct tree {
    int numara;
    char soyad[20];
    char adi[20];
    unsigned long int sayac;
    unsigned long int sol;
    unsigned long int sag;
} mem_kayit,temp,secondtemp,ayit,kayit = { 0,"                ","                ",
"         ","         ","    ","         ","         ",
"         ","         ","    ","         ","         " };

clock_t  t_ilk,t_son,zaman;
```

```
unsigned long int mem_kok,b1,b2;        /* First  Node in  Tree        */
unsigned long int kayityeri;
/*************************          ******************************/
struct  file_bilgisi {
unsigned long int kok;
unsigned long int ilkyer;    /* First  node in delete chain */
unsigned long int kayit_say; /* Dosyada kayit varmi  yokmu  */
}mem_key,key;
unsigned long int balance(struct tree mem_kayit,unsigned long int mem_kok);

FILE *keyfile,*data,*dosyaad,*dosyanumara,*kaycikti,*bulcikti;

unsigned long filesize(FILE  *stream)   {
unsigned long curpos, length;


          curpos = ftell(stream);
          fseek(stream, 0L, SEEK_END);
          length = ftell(stream);
          fseek(stream, curpos, SEEK_SET);
          return length;
           }

void  data_file_ac()  {
    if((data  = fopen("IPLVERI.DAT","r+b")) == NULL)
          {     data=fopen("IPLVERI.DAT","w+b"); }
           }




void  key_file_ac(){
    if((keyfile = fopen("IPLKEY.DAT","r+b")) == NULL) {
          keyfile = fopen("IPLKEY.DAT","w+b");      }

    if (filesize(keyfile)==0L) {
          key.kok=0; mem_kok=0;
          key.ilkyer=0;
          key.kayit_say=0;
          fseek(keyfile,0L,SEEK_SET);
          fwrite(&key,sizeof(key),1,keyfile);
          }
    else if (filesize(keyfile)!=0L) {
          fseek(keyfile,0L,SEEK_SET);
          fread(&key,sizeof(key),1,keyfile); mem_kok=key.kok;  }
          }
```

```
/********************** STORE TREE FUNCTION
**********************/
unsigned long  int stree(unsigned long int mem_kok,unsigned long int  kayityeri,int
numara){
        struct  tree  mem_kayit;
        fseek(data,sizeof(kayit)*(mem_kok-1),SEEK_SET);
        fread(&mem_kayit,sizeof(kayit),1,data );
        mem_kayit.sayac++;
    if(kayit.numara < mem_kayit.numara) {
     if(mem_kayit.sol==0){
        mem_kayit.sol=kayityeri;
        kayit.sag=0;
        kayit.sol=0;
        kayit.sayac=1;
        fseek(data,sizeof(kayit)*(mem_kok-1),SEEK_SET);
        fwrite(&mem_kayit,sizeof(kayit),1,data );
        fseek(data ,sizeof(kayit)*(kayityeri-1),SEEK_SET);
        fwrite(&kayit,sizeof(kayit),1,data );/* return(kayit);*/
        return(mem_kok);
        }
    else  if(mem_kayit.sol!=0){
        mem_kayit.sol=stree(mem_kayit.sol,kayityeri,numara);
        fseek(data ,sizeof(kayit)*(mem_kok-1),SEEK_SET);
        fwrite(&mem_kayit,sizeof(kayit),1,data );
        return(balance(mem_kayit,mem_kok));
        } }
  else  if(kayit.numara >= mem_kayit.numara) {
    if(mem_kayit.sag==0){
        mem_kayit.sag=kayityeri;
        kayit.sol=0;
        kayit.sag=0;



        kayit.sayac=1;
        fseek(data,sizeof(kayit)*(mem_kok-1),SEEK_SET);
        fwrite(&mem_kayit,sizeof(kayit),1,data );
        fseek(data ,sizeof(kayit)*(kayityeri-1),SEEK_SET);
        fwrite(&kayit,sizeof(kayit),1,data );/*  return(kayit);*/
        return(mem_kok);}
   else  if(mem_kayit.sag!=0){
        mem_kayit.sag=stree(mem_kayit.sag,kayityeri,numara);
        fseek(data ,sizeof(kayit)*(mem_kok-1),SEEK_SET);/* Bir alt nod'da  herhangi
bir balance islemi dengeleme islemi olmus ise alt node degisebilir.*/
        fwrite(&mem_kayit,sizeof(kayit),1,data );
        return(balance(mem_kayit,mem_kok));
        }                   }
        return(mem_kok);
```

```
        }

unsigned long int balance(struct tree mem_kayit,unsigned long int mem_kok){
        unsigned long int
mem_sag,mem_sol,kay_sol,kay_sag,kayit_yeri,temp_yeri,donensol,donensag;
        struct tree kayit,temp;
if(mem_kayit.sol==0 && mem_kayit.sag==0){
        return(mem_kok); }
   else if(mem_kayit.sol!=0L){
        fseek(data,sizeof(kayit)*(mem_kayit.sol-1),SEEK_SET);
        fread(&kayit,sizeof(kayit),1,data);
        kayit_yeri=mem_kayit.sol;
        mem_sol=kayit.sayac;
        mem_sag=mem_kayit.sayac-kayit.sayac-1;
   if(mem_sol< mem_sag ) {
        fseek(data,sizeof(kayit)*(mem_kayit.sag-1),SEEK_SET);
        fread(&kayit,sizeof(kayit),1,data);
        kayit_yeri=mem_kayit.sag;
        mem_sag=kayit.sayac;
        mem_sol=mem_kayit.sayac-kayit.sayac-1;}
        }
   else if(mem_kayit.sol==0L) {
        fseek(data,sizeof(kayit)*(mem_kayit.sag-1),SEEK_SET);
        fread(&kayit,sizeof(kayit),1,data);
        kayit_yeri=mem_kayit.sag;
        mem_sag=mem_kayit.sayac-1;
        mem_sol=0;
        }
 if (mem_sol==mem_sag) { return(mem_kok); /* Rotation Yok */ }
 if (mem_sol>mem_sag) {   /* Sol agirlik ML tree */
  if(kayit.sol==0L && kayit.sag==0L){return(mem_kok);}
  if(kayit.sag!=0L){
        fseek(data,sizeof(kayit)*(kayit.sag-1),SEEK_SET);



        fread(&temp,sizeof(kayit),1,data);
        temp_yeri=kayit.sag;
        kay_sag=temp.sayac;
        kay_sol=kayit.sayac-temp.sayac-1;
        }
     else if(kayit.sag==0L){
        fseek(data,sizeof(kayit)*(kayit.sol-1),SEEK_SET);
        fread(&temp,sizeof(kayit),1,data);
        temp_yeri=kayit.sol;
        kay_sol=temp.sayac;
        kay_sag=kayit.sayac-temp.sayac-1;}
```

```
if(kay_sol > mem_sag){  /* Tek  rotation  */
     sola1++;
     mem_kayit.sayac=mem_sag + kay_sag + 1 ;
     kayit.sayac=mem_kayit.sayac + kay_sol + 1 ;
     mem_kayit.sol=kayit.sag;
     kayit.sag=mem_kok;
     fseek(data,sizeof(kayit)*(mem_kok-1),SEEK_SET);
     fwrite(&mem_kayit,sizeof(kayit),1,data);
     fseek(data,sizeof(kayit)*(kayit_yeri-1),SEEK_SET);
     fwrite(&kayit,sizeof(kayit),1,data);
     donensag=balance(mem_kayit,mem_kok);
if(donensag) {
     kayit.sag=donensag;

     fseek(data,sizeof(kayit)*(kayit_yeri-1),SEEK_SET);
     fwrite(&kayit,sizeof(kayit),1,data); }
     return(kayit_yeri);
     }
else  if((kayit.sayac-kay_sol-1) > mem_sag) { /* iki  rotasyon */

     fseek(data,sizeof(kayit)*(kayit.sag-1),SEEK_SET);
     fread(&temp,sizeof(kayit),1,data);
     temp_yeri=kayit.sag;
if (temp.sol==0 &&  temp.sag==0){
     b1=0;
     b2=0;
     }
else  if (temp.sol!=0L) { /*  solunda  node  var */
     fseek(data,sizeof(kayit)*(temp.sol-1),SEEK_SET);
     fread(&secondtemp,sizeof(kayit),1,data);
     b1=secondtemp.sayac;
     b2=temp.sayac-secondtemp.sayac-1;
     }
else  if(temp.sag!=0L) {  /* saginda  node  var */
     fseek(data,sizeof(kayit)*(temp.sag-1),SEEK_SET);



     fread(&secondtemp,sizeof(kayit),1,data);
     b2=secondtemp.sayac;
     b1=temp.sayac-secondtemp.sayac-1;
     }
     kayit.sayac=b1+kay_sol+1;
     mem_kayit.sayac=b2 + mem_sag+1;
     temp.sayac = kayit.sayac + mem_kayit.sayac+1;
     kayit.sag=temp.sol;
```

```
        temp.sol=kayit_yeri;
        mem_kayit.sol=temp.sag;
        temp.sag=mem_kok;
        sola2++;
        fseek(data,sizeof(kayit)*(temp_yeri-1),SEEK_SET);
        fwrite(&temp,sizeof(kayit),1,data);
        fseek(data,sizeof(kayit)*(kayit_yeri-1),SEEK_SET);
        fwrite(&kayit,sizeof(kayit),1,data);
        fseek(data,sizeof(kayit)*(mem_kok-1),SEEK_SET);
        fwrite(&mem_kayit,sizeof(kayit),1,data);
        donensag=balance(mem_kayit,mem_kok);
        donensol=balance(kayit,kayit_yeri);
    if(temp.sol!=donensol || temp.sag!=donensag) {
        temp.sol=donensol;
        temp.sag=donensag;

        fseek(data,sizeof(kayit)*(temp_yeri-1),SEEK_SET);
        fwrite(&temp,sizeof(kayit),1,data);         }
        return(temp_yeri);  }
     else  {        return(mem_kok);  /* Rotation yok    */  }
        }
else  if (mem_sol < mem_sag)  {      /* Sag agirlik MR  */
    if(kayit.sol==0L && kayit.sag==0L){return(mem_kok);}
    if(kayit.sol!=0L){
        fseek(data,sizeof(kayit)*(kayit.sol-1),SEEK_SET);
        fread(&temp,sizeof(kayit),1,data);
        temp_yeri=kayit.sol;
        kay_sol=temp.sayac;
        kay_sag=kayit.sayac-temp.sayac-1;}
    else if (kayit.sag!=0L){
        fseek(data,sizeof(kayit)*(kayit.sag-1),SEEK_SET);
        fread(&temp,sizeof(kayit),1,data);
        temp_yeri=kayit.sag;
        kay_sag=temp.sayac;
        kay_sol=kayit.sayac-temp.sayac-1;
        }
    if(kay_sag>mem_sol){  /*  Tek  rotation  */
        saga1++;
        mem_kayit.sayac=mem_sol + kay_sol + 1 ;



        kayit.sayac=mem_kayit.sayac + kay_sag + 1 ;
        mem_kayit.sag=kayit.sol;
        kayit.sol=mem_kok;
        fseek(data,sizeof(kayit)*(mem_kok-1),SEEK_SET);
        fwrite(&mem_kayit,sizeof(kayit),1,data);
```

```
        fseek(data,sizeof(kayit)*(kayit_yeri-1),SEEK_SET);
        fwrite(&kayit,sizeof(kayit),1,data);
        donensol=balance(mem_kayit,mem_kok);
if(donensol!=kayit.sol){
        kayit.sol=donensol;
        fseek(data,sizeof(kayit)*(kayit_yeri-1),SEEK_SET);
        fwrite(&kayit,sizeof(kayit),1,data); }
        return(kayit_yeri);
        }

else  if((kayit.sayac-kay_sag-1) > mem_sol) { /* iki rotasyon */

        fseek(data,sizeof(kayit)*(kayit.sol-1),SEEK_SET);
        fread(&temp,sizeof(kayit),1,data);
        temp_yeri=kayit.sol;
        if (temp.sol==0 &&  temp.sag==0){
        b1=0;
        b2=0;
        }
else  if (temp.sol!=0L) { /*   solunda  node  var */
        fseek(data,sizeof(kayit)*(temp.sol-1),SEEK_SET);
        fread(&secondtemp,sizeof(kayit),1,data);
        b1=secondtemp.sayac;
        b2=temp.sayac-secondtemp.sayac-1;
        }
else  if(temp.sag!=0L) {  /* saginda  node  var */
        fseek(data,sizeof(kayit)*(temp.sag-1),SEEK_SET);
        fread(&secondtemp,sizeof(kayit),1,data);
        b2=secondtemp.sayac;
        b1=temp.sayac-secondtemp.sayac-1;
        }
        kayit.sayac=b2 + kay_sag+1;
        mem_kayit.sayac=b1 + mem_sol+1;
        temp.sayac = kayit.sayac + mem_kayit.sayac+1;
        kayit.sol=temp.sag;
        temp.sag=kayit_yeri;
        mem_kayit.sag=temp.sol;
        temp.sol=mem_kok;
        saga2++;
        fseek(data,sizeof(kayit)*(temp_yeri-1),SEEK_SET);
        fwrite(&temp,sizeof(kayit),1,data);
        fseek(data,sizeof(kayit)*(kayit_yeri-1),SEEK_SET);



        fwrite(&kayit,sizeof(kayit),1,data);
        fseek(data,sizeof(kayit)*(mem_kok-1),SEEK_SET);
```

```
        fwrite(&mem_kayit,sizeof(kayit),1,data);
        donensol=balance(mem_kayit,mem_kok);
        donensag=balance(kayit,kayit_yeri);
    if(temp.sol!=donensol || temp.sag!=donensag) {
        temp.sol=donensol;
        temp.sag=donensag;
        fseek(data,sizeof(kayit)*(temp_yeri-1),SEEK_SET);
        fwrite(&temp,sizeof(kayit),1,data); }
        return(temp_yeri);
        }
        else {      return(mem_kok); /* rotation yok */   } }
        }

unsigned long int stree_ilk() {
        unsigned long int donen_kok;
    if (key.kayit_say==0L) {
        key.ilkyer=0; key.kayit_say=1;
        key.kok=1; kayit.sol=0; kayit.sag=0;
        kayit.sayac=1;
        fseek(data,0L,SEEK_SET);

        fwrite(&kayit,sizeof(kayit),1,data);
        fseek(keyfile,0L,SEEK_SET);
        fwrite(&key,sizeof(key),1,keyfile);
        //fclose(data );fclose(keyfile);
        return(1); }
    if (key.ilkyer==0L) {
        kayityeri=key.kayit_say+1;
        key.kayit_say+=1;
        fseek(keyfile,0L,SEEK_SET);
        fwrite(&key,sizeof(key),1,keyfile); }

    else if (key.ilkyer!=0L) {
        kayityeri=key.ilkyer;
        fseek(data,sizeof(kayit)*(key.ilkyer-1),SEEK_SET);
        fread(&mem_kayit,sizeof(kayit),1,data );
        key.ilkyer=mem_kayit.sag;
        //key.kayit_say+=1;
        fseek(keyfile,0L,SEEK_SET);
        fwrite(&key,sizeof(key),1,keyfile);
        }
        donen_kok=stree(mem_kok,kayityeri,kayit.numara);
        if(key.kok!=donen_kok) {
        key.kok=donen_kok;
        fseek(keyfile,0L,SEEK_SET);
```

```
        fwrite(&key,sizeof(key),1,keyfile);}
        return(1);
        }

/************** DELETE NODE FONKLSIYONU ****************/

unsigned long  int dtree(unsigned long int mem_kok,int numara) {
        unsigned  long  int p,p2,eskisol,eskisag;
        struct  tree  mem_kayit,kayit; /*where wireless  tree  attached  to this  node  */
        fseek(data ,sizeof(mem_kayit)*(mem_kok-1),SEEK_SET);
        fread(&mem_kayit,sizeof(mem_kayit),1,data );
  if(kayit.numara == mem_kayit.numara)  {  /* delete  root */
       if(mem_kayit.sol==mem_kayit.sag) {  /* this  means  an  empty  tree */
        if(key.kayit_say==1L)  {
        key.kayit_say=0;
        key.ilkyer=0;//mem_kayit.sag=key.ilkyer;

        key.kok=0;
        }
        else  if(key.kayit_say!=1L)  {
        key.kayit_say-=1;  }
        mem_kayit.sag=key.ilkyer;
        key.ilkyer=mem_kok;
        fseek(data ,sizeof(mem_kayit)*(mem_kok-1),SEEK_SET);
        fwrite(&mem_kayit,sizeof(mem_kayit),1,data );
        fseek(keyfile,0L,SEEK_SET);
        fwrite(&key,sizeof(key),1,keyfile);
        return(0); }

  else  if(mem_kayit.sol==0L){  /*   sag  agac  varsa*/
        if(key.kok==mem_kok)
        key.kok=mem_kayit.sag;
        p=mem_kayit.sag;
        if(key.ilkyer==0)  mem_kayit.sag=0;
        else   mem_kayit.sag=key.ilkyer;
        key.ilkyer=mem_kok;
        key.kayit_say -=1;
        fseek(keyfile,0,SEEK_SET);
        fwrite(&key,sizeof(key),1,keyfile);
        fseek(data ,sizeof(mem_kayit)*(mem_kok-1),SEEK_SET);
        fwrite(&mem_kayit,sizeof(mem_kayit),1,data );
        //fclose(keyfile); fclose(data );
        return(p);  }
 else  if(mem_kayit.sag==0L){
        if (key.kok==mem_kok)
        key.kok=mem_kayit.sol;
```

```
        p=mem_kayit.sol;
        if(key.ilkyer==0L)  mem_kayit.sag=0;
        else   mem_kayit.sag=key.ilkyer;
        key.ilkyer=mem_kok;
        key.kayit_say -=1;
        fseek(keyfile,0,SEEK_SET);
        fwrite(&key,sizeof(key),1,keyfile);
        fseek(data ,sizeof(mem_kayit)*(mem_kok-1),SEEK_SET);
        fwrite(&mem_kayit,sizeof(mem_kayit),1,data );
        return(p);
                        }

         /********  or both tree present     *********/
 else{  /********  Sag daki node kok olacak  *********/
        if (key.kok==mem_kok)
        key.kok=mem_kayit.sag;
        p=mem_kayit.sag;
        p2=mem_kayit.sag;

        if(key.ilkyer==0) mem_kayit.sag=0;

        else  mem_kayit.sag=key.ilkyer;
        key.ilkyer=mem_kok;
        key.kayit_say -=1;
        fseek(data ,sizeof(mem_kayit)*(mem_kok-1),SEEK_SET);
        fwrite(&mem_kayit,sizeof(mem_kayit),1,data);
while(p!=0){
        fseek(data ,sizeof(kayit)*(p-1),SEEK_SET);
        fread(&kayit,sizeof(kayit),1,data );
        if (kayit.sol==0) break;
        else  p=kayit.sol;
        }
        kayit.sol=mem_kayit.sol;
        fseek(keyfile,0,SEEK_SET);
        fwrite(&key,sizeof(key),1,keyfile);
        fseek(data ,sizeof(mem_kayit)*(p-1),SEEK_SET);
        fwrite(&kayit,sizeof(mem_kayit),1,data);
        return(p2); }
        }  /* ilk if in  sonu  */
else if (kayit.numara > mem_kayit.numara) { eskisag=mem_kayit.sag;
        mem_kayit.sag=dtree(mem_kayit.sag,kayit.numara);
        if(eskisag!=mem_kayit.sag){
        fseek(data ,sizeof(mem_kayit)*(mem_kok-1),SEEK_SET);
        fwrite(&mem_kayit,sizeof(mem_kayit),1,data );}}
 else if(kayit.numara < mem_kayit.numara)  {  eskisol=mem_kayit.sol;
        mem_kayit.sol=dtree(mem_kayit.sol,kayit.numara);
        if(eskisol!=mem_kayit.sol){
```

```
        fseek(data ,sizeof(mem_kayit)*(mem_kok-1),SEEK_SET);



        fwrite(&mem_kayit,sizeof(mem_kayit),1,data ); }}
        return(mem_kok);
        }

/************* SEARCH TREE FUNCTION **************/
unsigned long int search_tree(unsigned long int mem_kok,int numara) {

 if  (!mem_kok) { return(0); }
        fseek(data ,sizeof(kayit)*(mem_kok-1),SEEK_SET);
        fread(&mem_kayit,sizeof(kayit),1,data );

 while (mem_kayit.numara != numara) {
 if   (numara < mem_kayit.numara)   mem_kok=mem_kayit.sol ;
 else  mem_kok=mem_kayit.sag;
  if  (mem_kok==0)  break;
        fseek(data ,sizeof(kayit)*(mem_kok-1),SEEK_SET);
        fread(&mem_kayit,sizeof(kayit),1,data );
        }
  if (mem_kayit.numara==numara) {  return(1); }

   else { return(0); }
                }

void data_dosyalarini_ac(){

  if ((dosyaad = fopen("adi.dat", "rt")) == NULL)
        { fprintf(stderr, "Cikti dosyasi acilamadi\n");  }

  if ((dosyanumara = fopen("numara.dat", "rb")) == NULL)
        { fprintf(stderr, "Cikti dosyasi acilamadi\n");  }
         }

void    data_dosyalarini_kapa(){
           fclose(dosyaad);
           fclose(dosyanumara);
}

void kaydet() {
        int i,j,STAT=0;
        data_dosyalarini_ac();
  if ((kaycikti = fopen("ckayit.dat", "wt")) == NULL)
        { fprintf(stderr, "Cikti dosyasi acilamadi\n");  }
        for(i=0; i<10; i++) {
        zaman=0;
        for(j=0; j< INTERVAL; j++) {
        fread(kayit.adi, 20, 1, dosyaad);
```

```
        fread(&kayit.numara, sizeof(int), 1, dosyanumara);



        key_file_ac(); data_file_ac();
        t_ilk = clock();
     STAT=stree_ilk();  // kaydi sakla
        t_son = clock();
        printf("sayyorum  %d\n",ii++);
        fclose(keyfile);
        fclose(data);
          if(STAT == 0)
                printf("Insert Etmede Hata Var!  %d",STAT);
        zaman += (t_son  - t_ilk) ;

        }
        fprintf(kaycikti,"\n- %d - ile - %d - arasindaki KAYIT icin gecen sure: %f\n", i *
INTERVAL,(i+1) *INTERVAL ,((zaman) / CLK_TCK)/INTERVAL);
        }
        data_dosyalarini_kapa();
        fclose(kaycikti);
}


void bulma(){
        int i,j,STAT=0;
        data_dosyalarini_ac();
  if ((bulcikti = fopen("cbulma.dat", "wt")) == NULL)
        { fprintf(stderr, "Cikti dosyasi acilamadi\n");  }
        for(i=0; i<10; i++) {
        zaman=0;
        fseek(dosyanumara,((long)((i)*((INTERVAL)*sizeof(int)))), SEEK_SET);
        for(j=0; j< INTERVAL; j++) {
        key_file_ac(); data_file_ac();
        fread(&kayit.numara, sizeof(int), 1, dosyanumara);
        stree_ilk();
        fclose(keyfile);
        fclose(data);
        }
        fseek(dosyanumara, 0L, SEEK_SET);
        for(j=0; j< (i+1)*INTERVAL; j++) {
        fread(&ayit.numara, sizeof(int), 1, dosyanumara);
        key_file_ac(); data_file_ac();
        t_ilk = clock();
        STAT=search_tree(key.kok,ayit.numara);
        t_son = clock();
          if(STAT==0)
```

```
            {  printf("\nAranan numara dosyada bulunamadi!!! ");  }
        zaman += (t_son  - t_ilk) ;
        fclose(keyfile);
        fclose(data);




                                        }
        fprintf(bulcikti,"\n-%d- ile -%d- arasindaki BULMA icin gecen sure:
%f\n",0,(i+1) *INTERVAL,((zaman) / CLK_TCK)/(INTERVAL*(i+1)));


        }  // ilk for sonu
        data_dosyalarini_kapa();
        fclose(bulcikti);
        printf("\nBulma icin gerekli islemler tamamlandi  ");
          }


main()
{       char komut[20];
        system("del iplveri.dat");

        system("del iplkey.dat");

        key_file_ac(); data_file_ac();

   for( ; ; )  {

        printf("\n  ****  Asagidaki komutlardan birini seciniz  **** \n");
        printf("1-Sicil kaydi GIRME\n");
        printf("2-Sicil kaydi BULMA\n");
        printf("3-Cikis\n");
        printf("Bir komut giriniz( 1 ile 3 arasinda)\n");
        gets(komut);
        switch(komut[0]) {
                case '1': kaydet();  break;
                case '2': bulma();  break;
                case '3': fclose(keyfile);fclose(data); exit(0);
         default :  printf("Yanlis komut secildi!!!");

                }
}
 }
```

4. TABLOLAR ve GRAFİKLER

4.1. KAYIT GİRİŞİ

BSİCİL.C, SİCİL.C ve CSİCIL.C programlarının çalıştırılması ile elde edilen kayıt giriş sürelerine ait değerler aşağıdaki tabloda toplu olarak verilmiştir:

Kayıt sayısı	BTRIEVE	DB - VISTA	IPR
400.	0.207280	0.007230	0.106863
- 800.	0.209066	0.013736	0.151099
- 1200.	0.209615	0.019368	0.176099
- 1600.	0.209066	0.020467	0.193132
- 2000.	0.213874	0.022390	0.208791
- 2400.	0.212363	0.025275	0.216209
- 2800.	0.233929	0.026236	0.227335
- 3200.	0.260989	0.026786	0.234890
- 3600.	0.259478	0.027335	0.241071
- 4000.	0.253159	0.027610	0.244505

Tablo 1. Alfasayısal anahtar için

BSİCİLS.C, SİCLS.C ve CSİCİLS.C programlarının çalıştırılması ile elde edilen kayıt giriş sürelerine ait değerler aşağıdaki tabloda toplu olarak verilmiştir :

Kayıt sayısı	BTRIEVE	DB - VISTA	IPR
400.	0.169505	0.005907	0.112637
- 800.	0.170330	0.008242	0.165110
- 1200.	0.181593	0.012363	0.191346
- 1600.	0.195055	0.015797	0.210302
- 2000.	0.201511	0.017445	0.221566
- 2400.	0.224451	0.017445	0.234341
- 2800.	0.232967	0.017995	0.236676
- 3200.	0.237225	0.019918	0.247390
- 3600.	0.246703	0.019363	0.254396
- 4000.	0.251236	0.021841	0.261264

Tablo 2. Sayısal anahtar için

4.2. KAYIT OKUMA

BSİCİL.C, SİCİL.C ve CSİCİL.C programlarından elde edilen kayıt okuma sürelerine ait değerler aşağıdaki tabloda toplu olarak verilmiştir:

Kayıt sayısı	BTRIEVE	DB-VISTA	IPR
400	0.001923	0.001099	0.016621
800	0.001717	0.001854	0.018887
1200	0.001694	0.002015	0.020330
1600	0.001648	0.001957	0.021738
2000	0.001621	0.002005	0.022198
2400	0.001694	0.002175	0.021932
2800	0.001609	0.002041	0.022038
3200	0.001666	0.002284	0.023472
3600	0.001679	0.002473	0.023092
4000	0.001621	0.002486	0.023475

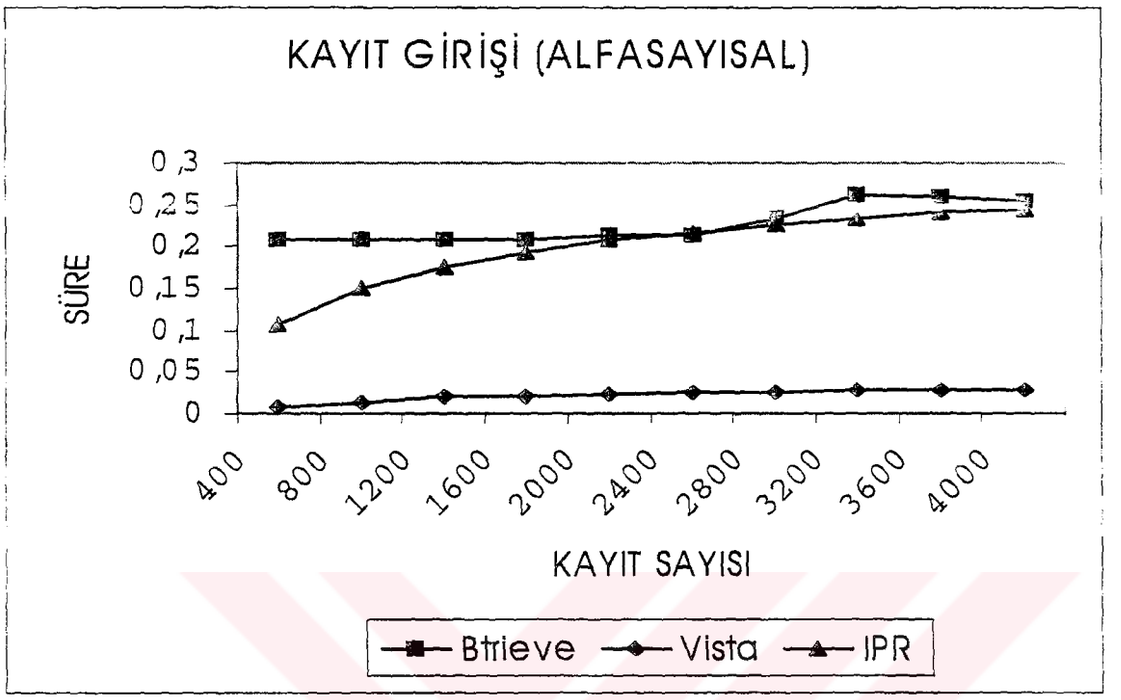
Tablo 3. Alfasayısal anahtar için

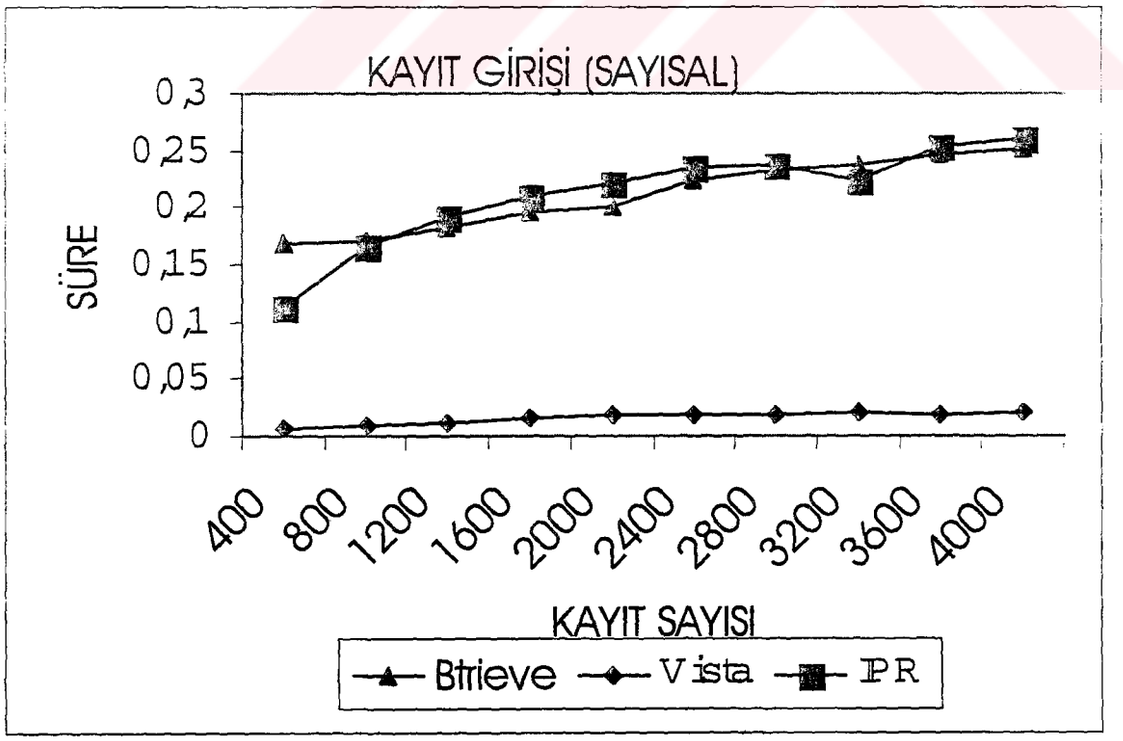
BSİCİLS.C, SİCİLS.C ve CSİCİLS.C programlarından elde edilen kayıt okuma süreleri aşağıdaki tabloda toplu olarak verilmiştir :

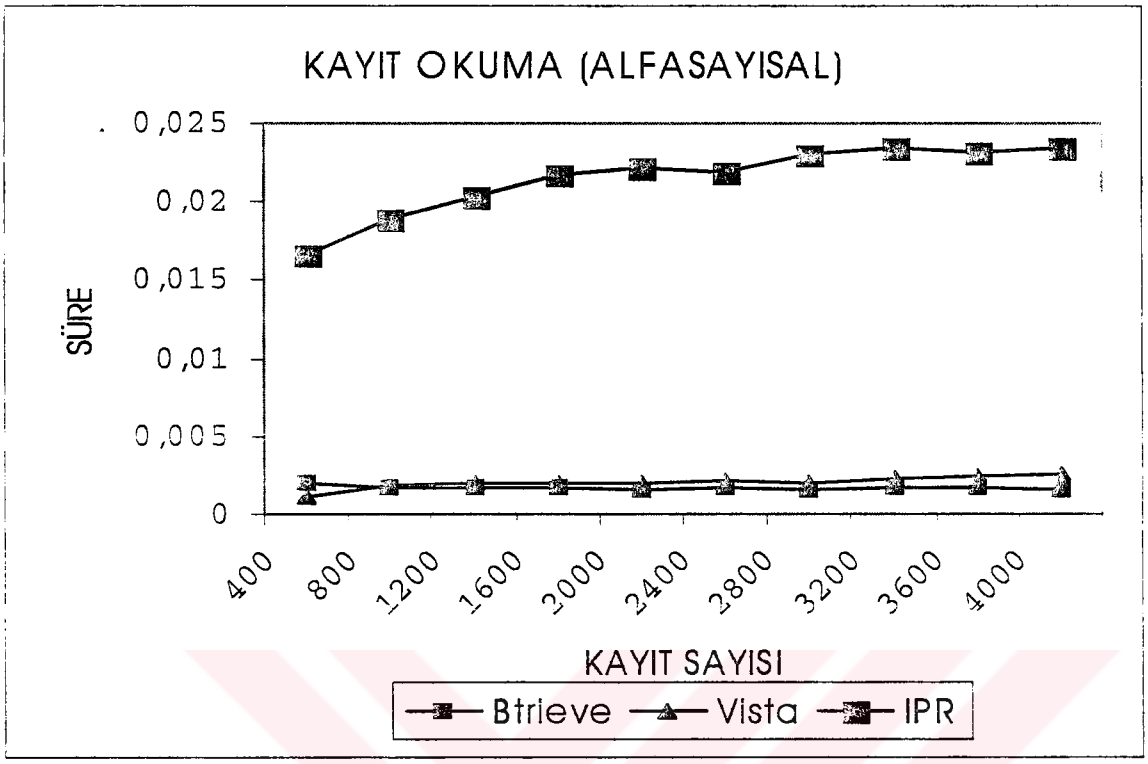
Kayıt sayısı	BTRIEVE	DB-VISTA	IPR
400	0.002885	0.000687	0.013187
800	0.002679	0.000962	0.017102
1200	0.003068	0.001374	0.017628
1600	0.003159	0.001374	0.019574
2000	0.003462	0.001566	0.020055
2400	0.003686	0.001580	0.020353
2800	0.003807	0.001668	0.020918
3200	0.003966	0.001734	0.022424
3600	0.004121	0.001694	0.022863
4000	0.004299	0.001813	0.023077

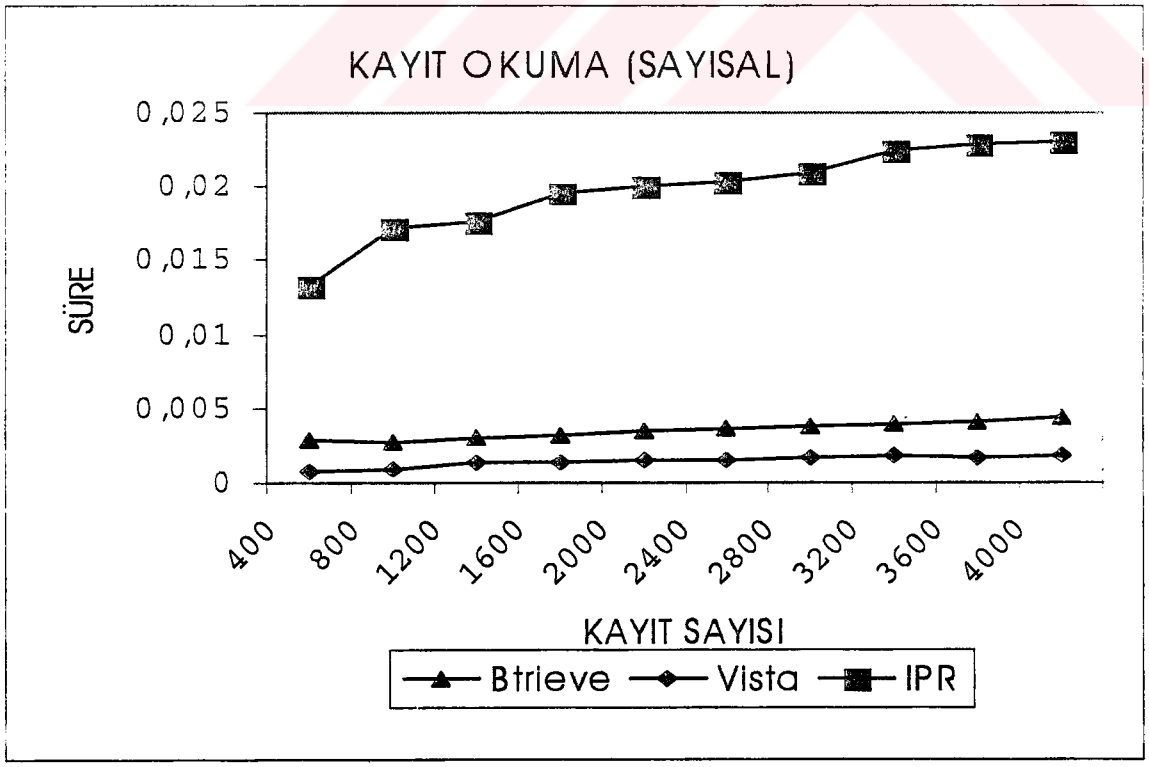
Tablo 4. Sayısal anahtar için

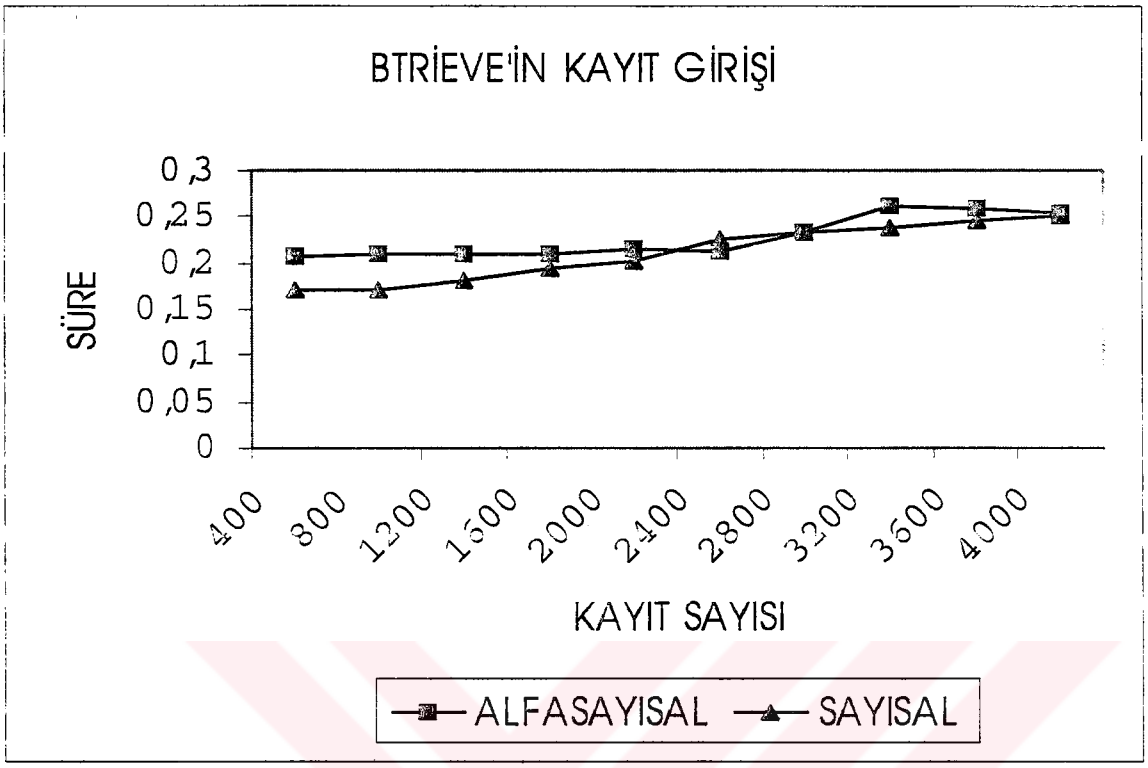
4.3. GRAFİKLER

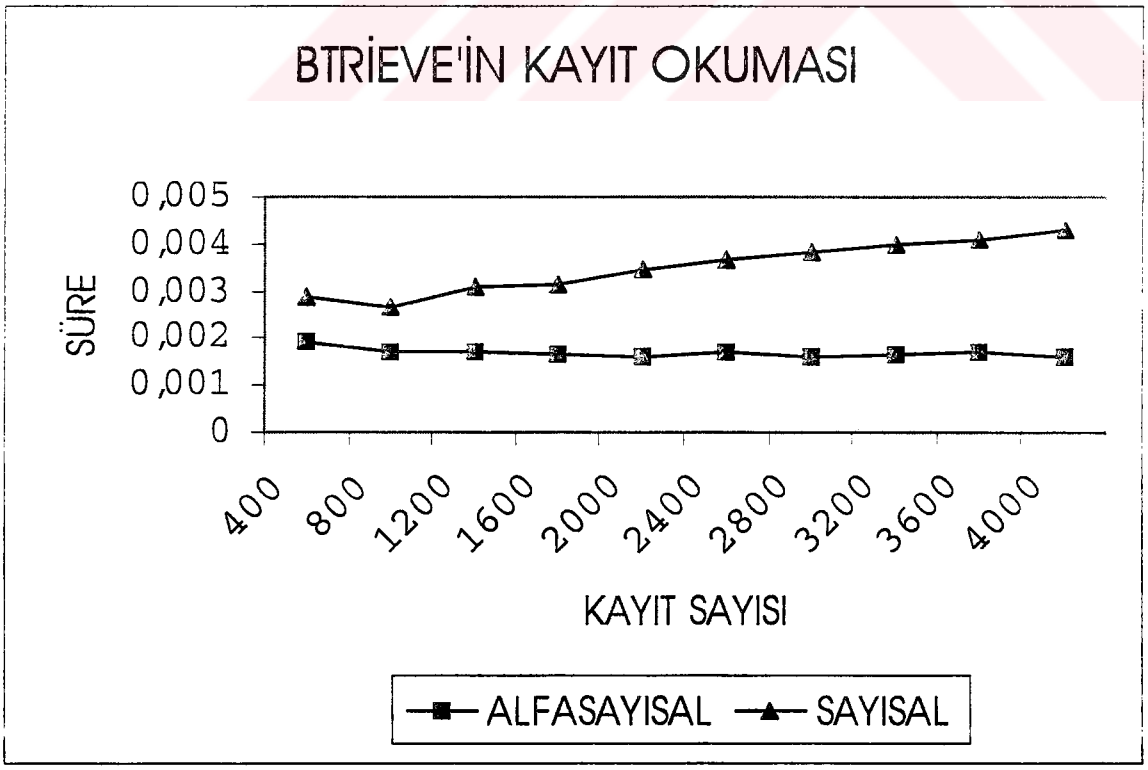


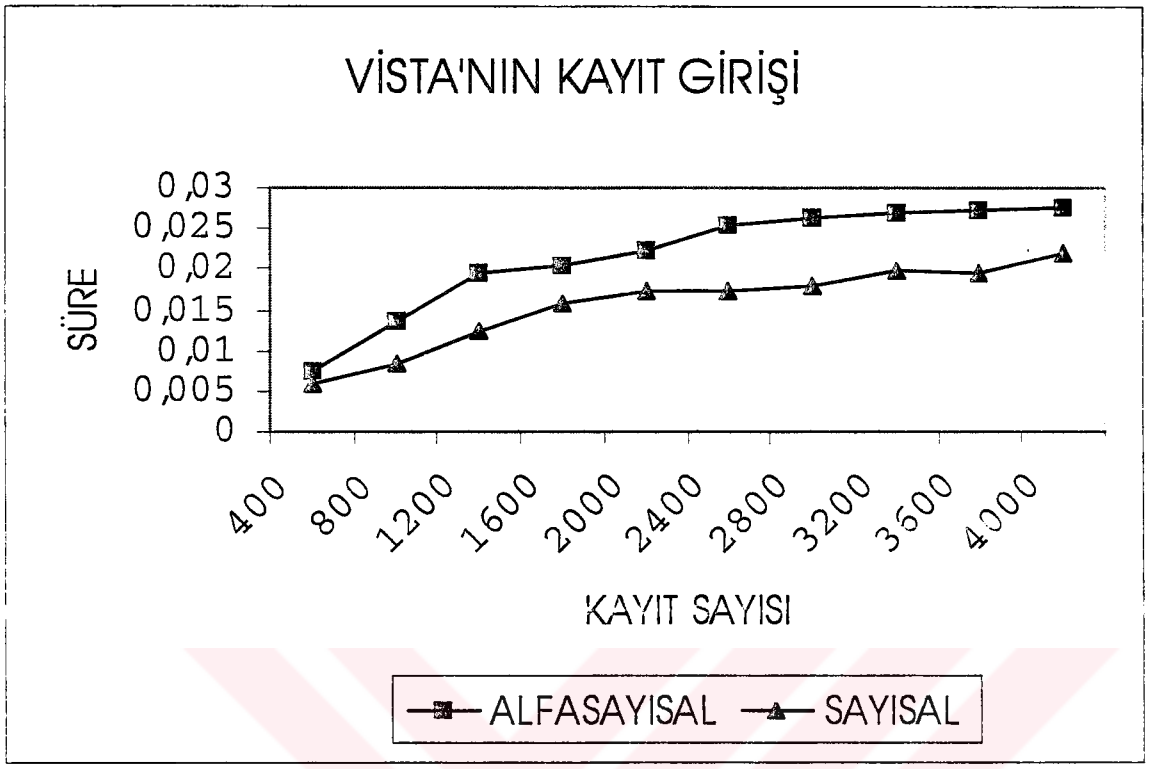


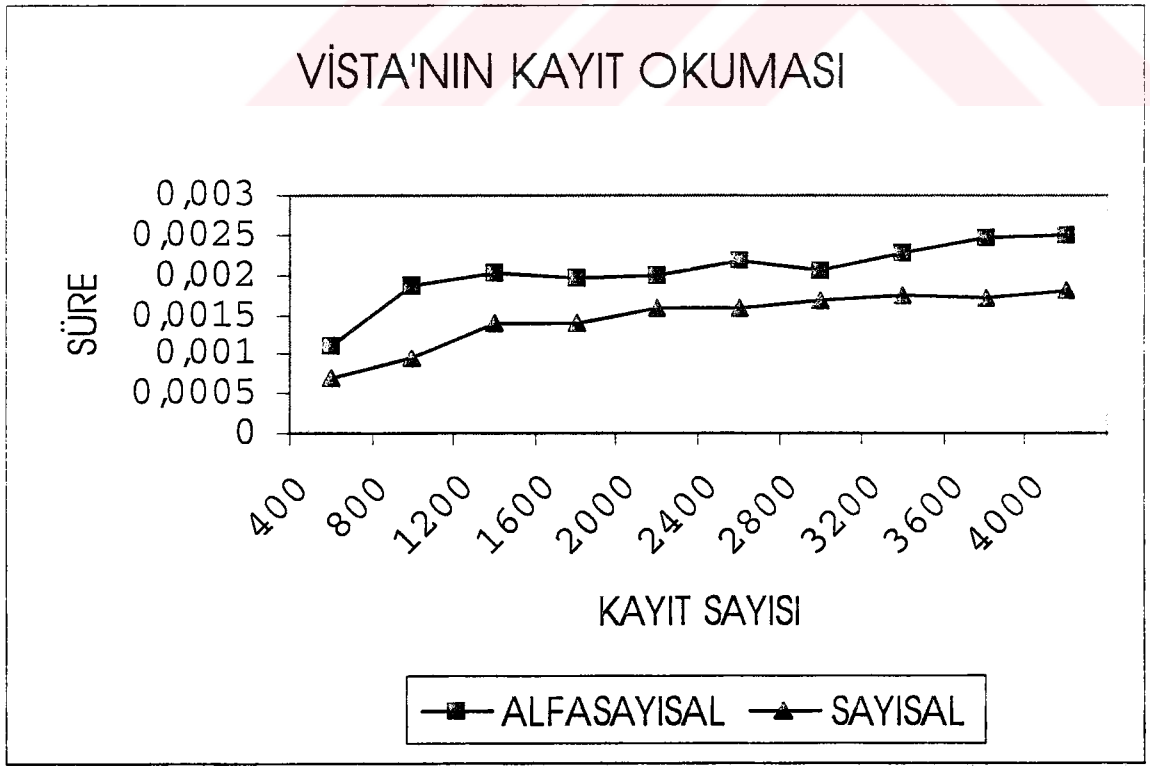

KAYIT OKUMA (ALFASAYISAL)
SÜRE
0,025
0,02
0,015
0,01
0,005
0
400
800
1200
1600
2000
2400
2800
3200
3600
4000
KAYIT SAYISI
Btrieve
Vista
IPR

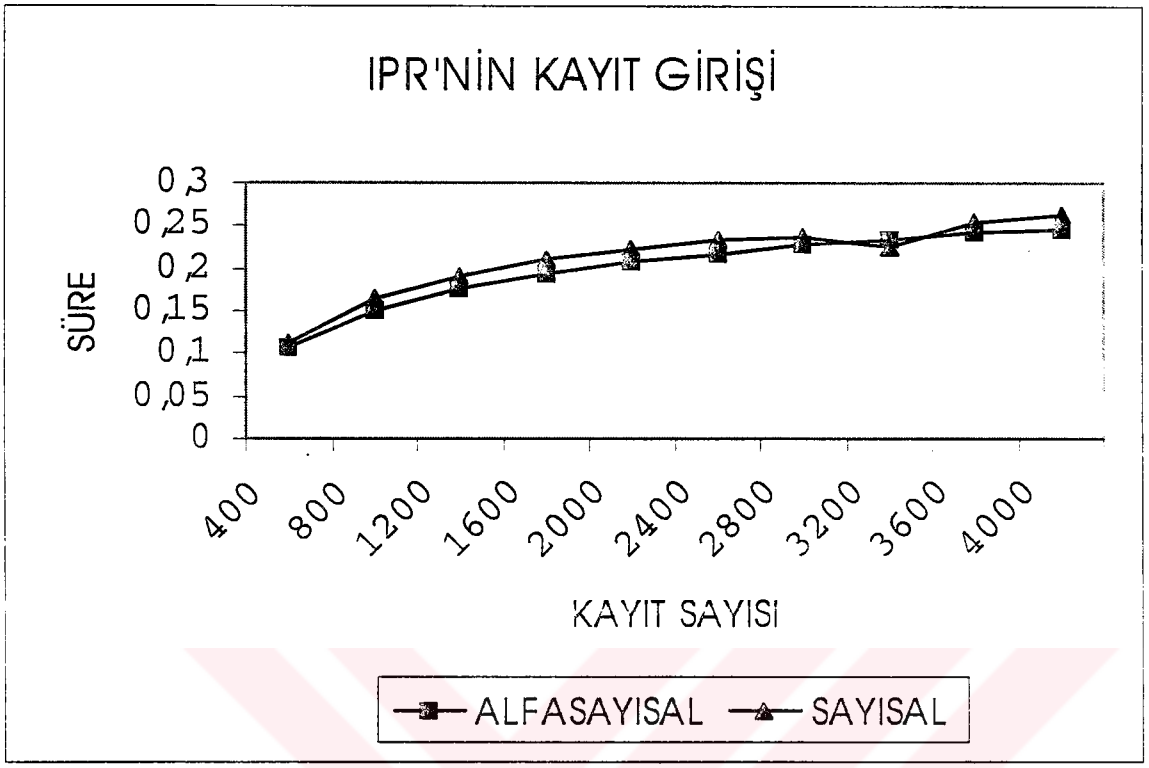

KAYIT OKUMA (SAYISAL)
SÜRE
0,025
0,02
0,015
0,01
0,005
0
400
800
1200
1600
2000
2400
2800
3200
3600
4000
KAYIT SAYISI
Btrieve
Vista
IPR

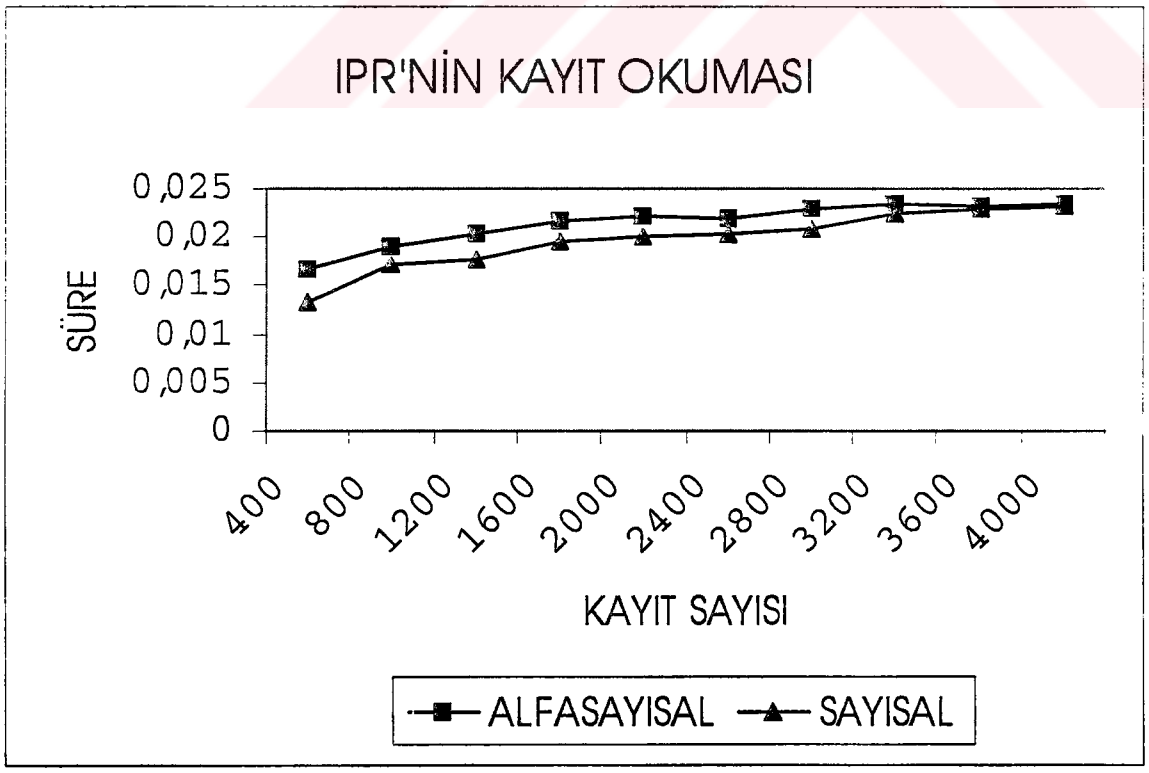

BTRİEVE'İN KAYIT GİRİŞİ
SÜRE
0,3
0,25
0,2
0,15
0,1
0,05
0
400
800
1200
1600
2000
2400
2800
3200
3600
4000
KAYIT SAYISI
ALFASAYISAL
SAYISAL


BTRİEVE'İN KAYIT OKUMASI
SÜRE
0,005
0,004
0,003
0,002
0,001
0
400
800
1200
1600
2000
2400
2800
3200
3600
4000
KAYIT SAYISI
ALFASAYISAL
SAYISAL


VİSTA'NIN KAYIT GİRİŞİ
SÜRE
0,03
0,025
0,02
0,015
0,01
0,005
0
400
800
1200
1600
2000
2400
2800
3200
3600
4000
KAYIT SAYISI
ALFASAYISAL
SAYISAL


VİSTA'NIN KAYIT OKUMASI
SÜRE
0,003
0,0025
0,002
0,0015
0,001
0,0005
0
400
800
1200
1600
2000
2400
2800
3200
3600
4000
KAYIT SAYISI
ALFASAYISAL
SAYISAL


IPR'NİN KAYIT GİRİŞİ
SÜRE
0,3
0,25
0,2
0,15
0,1
0,05
0
400
800
1200
1600
2000
2400
2800
3200
3600
4000
KAYIT SAYISI
ALFASAYISAL
SAYISAL


IPR'NİN KAYIT OKUMASI
SÜRE
0,025
0,02
0,015
0,01
0,005
0
400
800
1200
1600
2000
2400
2800
3200
3600
4000
KAYIT SAYISI
ALFASAYISAL
SAYISAL

5. SONUÇ VE TEZE İLİŞKİN ÖNERİLER

5.1. SONUÇ

SİCİL.C, BSİCİL.C ve CSİCİL.C programlarında, 20'şer karakterlik ad ve soyad alanlarının bulunduğu 4000 adet kaydın dosyaya yazılması ve dosyadan okunması işlemleri 400'er kayıt ara ile yapılmaktadır. Bu aralardaki ortalama süre hesaplanıp tek bir kayıt için yaklaşık süre, üç database ortamında da elde edilip tablolarda gösterilmektedir. Kayıt okuma ve yazma işlemlerinde soyad anahtar alan olarak seçilip, bu alanlardaki kayıtlar tekrarlanmamaktadır.

Kayıt giriş işlemlerinde;

Db-Vista / Btrieve =0.095

Db-Vista / IPR = 0.108

Btrieve / IPR = 1.134

oranları bulunmaktadır. Bu oranlar dikkate alındığında, dosyaya yazmada Db-Vista'nın çok hızlı olduğu, diğer ikisinin birbirine yakın olduğu gözlenmektedir.

Kayıt okuma işlemlerinde;

Db-Vista / Btrieve = 1.079

Db-Vista / IPR = 0.008

Btrieve / IPR =0.078

oranları bulunmaktadır. Bu oranlara göre, dosyadan kayıt okuma işlemini Db-Vista ve Btrieve hemen hemen eşit sürede yaparken, IPR en yavaş okumayı yapmıştır.

SİCİLS.C, BSİCİLS.C ve CSİCİLS.C programlarında dosyaya kayıt yazma ve dosyadan kayıt okuma işlemleri, aynen SİCİL.C, BSİCİL.C ve CSİCİL.C programlarında olduğu gibi gerçekleştirilmiştir. Burada dosya alanları numara ve ad alanlarından oluşmaktadır. Ad 20 karakterlik string alan olarak seçilmekte; numara alanı esas anahtar olup değeri 0 ile 65.000 arasında değişen random interger sayılardan seçilmektedir. Bu programlar 4000 kayıt için incelenmektedir.

Kayıt giriş işlemlerinde;
Db-Vista / Btrieve = 0.074
Db-Vista / IPR = 0.073
Btrieve / IPR = 0.988

oranları bulunmaktadır. Bu oranlar dikkate alındığında, dosyaya yazmada Db-Vista'nın en hızlı olduğu, Btrieve'in IPR'ye göre biraz daha hızlı olduğu gözlenmektedir.

Kayıt okuma işlemlerinde;
Db-Vista / Btrieve = 0.411
Db-Vista / IPR = 0.073
Btrieve / IPR = 0.178

oranları bulunmuştur. Bu oranlara göre, kayıt okumada Db-Vista'nın en hızlı olduğu, IPR'nin ise en yavaş olduğu gözlenmektedir.

Tüm bu sonuçlara göre, alfasayısal ve sayısal anahtar alanlarının seçili olduğu her durumda Db-Vista'nın kayıt okuma ve yazma işlemlerinde en hızlı, IPR yönteminin ise en yavaş olduğu sonucu çıkarılabilir.

5.2. ÖNERİLER

Tezde sadece kayıt giriş ve okuma işlemleri tekrarlanmayan tek bir anahtar alan için (sayısal ve alfasayısal değişkenlere) uygulanmaktadır. Bu tezin devamı olarak yapılacak çalışmalarda;

- Tek bir anahtar alan için, tekrarlanan kayıtlar
- Birden fazla anahtar için tekrarlanmayan ve tekrarlanan kayıtlar

incelenebilir.

Kayıt düzeltme işlemi kayıt giriş ve kayıt okuma işlemlerinin toplamı olarak bilindiğinden, aynı datalar üç database ortamında, kayıt düzeltme fonksiyonları için çalıştırılıp kayıt düzeltme süresi hesaplanabilir. Hesaplanan bu süre ile kayıt okuma ve kayıt giriş sürelerinin toplamı kıyaslanabilir.

KAYNAKLAR

1. Bayburan B, Microsoft Standart C, Beta Basım Yayım A.Ş., İstanbul, 1991

2. Internet http://www.Btrieve.com adresi

3. Joan B., File Structures, Salezberg, 1988

4. Raima Corporation, Raima Data Manager Reference Manual, Bellevue, U.S.A.

5. Raima Corporation, Raima Data Manager User's Guide, Bellevue, U.S.A.

6.SoftCraft, Inc. (Novell Development Divsion), Btrieve Reference Manual

7. Thur L. Alan, File Orgnization and Processing, North Caroline State University, 1988

ÖZGEÇMİŞ

03.05.1973 tarihinde Silivri'de doğdum. İlk, orta ve lise tahsilimi Silivri'de tamamladım. 1991'de Trakya Üniversitesi Mühendislik-Mimarlık Fakültesi Bilgisayar Mühendisliği Bölümü'nü kazandım. 4 yıl lisans eğitiminden sonra 1995 yılında Bilgisayar Mühendisi olarak mezun oldum. Eylül 1995'te, mezun olduğum bölümde Yüksek Lisans programına girmeye hak kazandım. Aynı yıl, Trakya Üniversitesi Eğitim Fakültesi'nde Araştırma Görevlisi olarak çalışmaya başladım. Halen bu fakültede Öğretim Görevlisi olarak hizmet etmekteyim.