



YILDIZ TEKNİK ÜNİVERSİTESİ  
ELEKTRİK-ELEKTRONİK FAKÜLTESİ  
BİLGİSAYAR MÜHENDİSLİĞİ  
2014-2015 ÖĞRETİM YILI GÜZ YARIYILI

# VERİ YAPILARI VE ALGORİTMALAR ÖDEV-1 ( BLM-2512/ GRUP:1)

**Hazırlanan Anabilim Dalı**  
**Bilgisayar Bilimleri Anabilim Dalı**

**Hazırlayan**  
Mert Sevil  
09013057  
Bilgisayar Mühendisliği Lisans Programı

**Öğretim Üyesi**  
Prof. Dr. M. Yahya KARSLIGİL

**İSTANBUL, 2014**

## İçindekiler

1. Ödevin amacı, tanıtımı ve giriş.....	2,3
2. Veri yapıları ve çeşitleri.....	3
3. Ağaç veri yapısı .....	4
3.1. Çeşitleri.....	4
3.2. Ağaca veri ekleme.....	5
3.3. Ağaçtan veri silme.....	5
3.4. İkili arama ağacı.....	6
3.5. Ağaç üzerinde eleman arama.....	7
3.6. Avantajları.....	7
4. Ödevin gerçekleştirilmesi.....	8
4.1. Kodun yazılması.....	8,9,10,11
4.2. Çıktıların elde edilmesi ve algoritma analizi.....	12,13,14
4.3. Sonuçların yorumlanması.....	14
5. Kaynakça.....	14

### 1. Ödevin amacı, tanıtımı ve giriş

#### Ödevin amacı:

- ✓ Veri yapılarını tanımak ve teorik olarak basitçe incelemek
- ✓ Ağaç verisi yapısını detaylarıyla incelemek
- ✓ Ağaçlarda eleman eklemesi ve arama algoritmalarının kodlanması
- ✓ Programlama dillerinden C dili ile kod geliştirme yeteneğinin geliştirilmesi
- ✓ Gereksiz döngülerden kaçınarak daha sadece ve kullanışlı algoritmalarının tasarlanabilmesi ve gerçekleştirilebilmesi
- ✓ Menü ile kullanıcı odaklı kod yazabilme yeteneğinin oluşturulması
- ✓ Farklı arama algoritmalarını tanımak ve aralarındaki farkları gözlemleyebilmek

Bu açıdan ödev de C programlama dili ile verilen algoritmalar gerçekleştirilmiştir. Sonuçları analiz edilmiş ve sonuçlar bölümünde tartışılmıştır. Kodun gerçekleştirme bölümü dışında detaylı bir ön teorik araştırma ile konu kavranmaya çalışılmış ve öğrenilen bilgiler ödevle özetlenmiştir.

## Giriş

Veri yapısı, bilgisayar ortamında verilerin etkin olarak saklanması ve işlenmesi için kullanılan yapı.[1]

Veri yapıları, verilerin düzenlenme biçimini belirleyen yapıtaşlarıdır. Bir yazılım değişkeni bile basit bir veri yapısı olarak kabul edilebilir. Değişik algoritmalarda verilerin diziler, listeler, yığınlar, kuyruklar, ağaçlar ve çizgeler gibi veri modellerine uydurularak düzenlenmesi gerekebilir. Veri, yapı ve algoritma bir yazılımın birbirinden ayrılmaz bileşenleridir. Algoritması hazırlanmış her yapı için verilerin düzenli bir şekilde kullanımı önemlidir. Çünkü yapı iyi kurulduğunda, etkin, doğru, anlaşılır ve hızlı çalışıp az kaynak kullanan algoritma geliştirmek kolaylaşır.[1]

## 2. Veri yapıları ve çeşitleri

Bir önceki bölümde tanımlanan veri yapıları algoritma dizaynında oldukça önemlidir. Doğru veri yapısının seçimi ve en uygun şekilde gerçekleştirilmesi özellikle hafıza ve hız önemi olan gerçek zamanlı bilgisayar uygulamaları veya gömülü sistem uygulamalarında önem kazanır.

Veri yapısı çeşitleri aşağıda listelenmiştir.

**Dizi:** En basit ve en sık kullanılan veri yapısıdır. Verileri adreslerine göre belli bir düzen içerisinde tutmak için kullanılır. Boyutlarına göre matris veya tensör olarak tanımlanıp farklı kullanım amaçlarına hizmet edebilirler. Ayrıca diziler farklı veri yapılarını tanımlamak içinde kullanılabilir. Örneğin çalışmamızda ağaç veri yapısı bir dizi olarak tanımlanmış olup, 2li ağaç algoritmasının getirdiği özellik ile dizi bir ağaç özelliği kazanmıştır.

**Linkli liste:** Bağlı liste herhangi bir tipten node'ların (düğümlerin) yine kendi tiplerinden düğümlere işaret etmesi (point) ile oluşan zincire verilen isimdir. Buna göre her düğümde kendi tipinden bir pointer olacak ve bu pointerlar ile düğümler birbirine aşağıdaki şekilde bağlanmaktadır. Linkli Liste'nin avantajı, hafızayı dinamik olarak kullanmasıdır. Buna göre hafızadan silinen bir bilgi için hafıza alanı boşaltılacak veya yeni eklenen bir bilgi için sadece o bilgiyi tutmaya yetecek kadar hafıza alanı ayrılacaktır. [2]

**Yığın ve Kuyruk:** Bu verisi yapısı günlük yaşamdaki gözlemlerin bilgisayara yansımasıdır. Bir banka gişesinde nasıl hizmet kuyruk sırasına göre yani ilk giren ilk çıkar (FIFO) mantığına göre gerçekleştiriliyorsa, bazı kavramlar kuyruk yapısı ile çözümlenmelidir. Yığın ise son gelenin ilk çıktığı sistemdir (LIFO). Ayrıca yığın veri yapısı bilgisayar mimarisi açısından büyük önem taşır. Çünkü bilgisayar arka planda işlemleri gerçekleştirirken bazı işlemleri geçici

süre için yığın alanında saklar. Bu açıdan yığın veri alanı bilgisayar organizasyonu açısından oldukça önemli bir konudur.

**Ağaç:** Ödevimiz olan ağaç veri yapısı 3. Bölümde detaylarıyla tartışılmıştır. Çok büyük verilerin bulunduğu sıralama algoritmalarında özellikle ikili arama ağacı olarak tanımlanmış veri yapılarının daha hızlı sonuçlar ürettiği gözlemlenmektedir. Ancak silme ve ağaçların dengeli tanımlanması sorunları temel dezavantajlar olarak görülmektedir.

**Graf:** Özellikle en kısa yol bulma gibi konularda popüler olarak kullanılan graf veri yapısından veriler ve komşuları arasındaki ilişkiler belirli düzenler içerisinde tanımlanır ve tutulur. Bu bilgiler uygun algoritmalar sayesinde anlamlandırılır. Ağaç veri yapısı kapalı bir göz oluşturmeyen graf veri yapısı olduğu için her ağaç verisi yapısı aynı zamanda bir graf veri yapısının özel bir halidir.

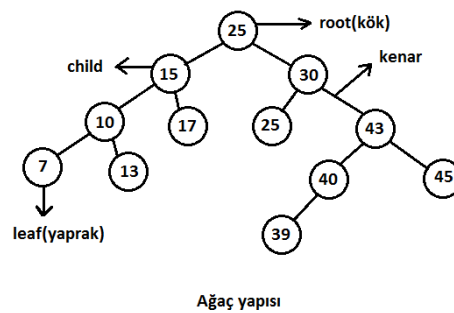
Sıklıkla kullanılan veri yapıları bu şekilde özetlenebilir. Böylece bu bölümde veri yapıları en basit tanımlarıyla incelenmiştir. Bölüm 3. te ödevimizin konusu olan ağaç veri yapısı ele alınmıştır.

### 3. Ağaç veri yapısı

Bu bölümde ağaç veri yapısı incelenmiştir. Buna göre önce ağaç yapısının özellikleri belirlenmiştir.

- ✓ Ağaçlar hiyerarşik ilişkileri göstermek için kullanılır.
- ✓ Her ağaç node'lar ve kenarlardan (edge) oluşur.
- ✓ Her bir node bir nesneyi gösterir.
- ✓ Her bir kenar (bağlantı) iki node arasındaki ilişkiyi gösterir.
- ✓ Arama işlemi bağlı dizilere göre çok hızlı yapılır. [3]

Örnek bir ağaç veri yapısı ve bu yapı üzerinde terminolojik tanımlamalar Şekil-1 de belirtilmiştir.



Şekil- 1 Örnek bir ağaç veri yapısı

### 3.1. Ağaç veri yapısının çeşitleri

En çok bilinen ağaç türleri ikili arama ağacı olup kodlama ağacı, sözlük ağacı, kümeleme ağacı gibi birçok ağaç uygulaması vardır.

- ✓ **B Ağacı:** B tree (B ağacı) binary (ikili) sıralama ağaçlarının genel halidir. Genel olarak arama işleminde daha hızlı sonuç vermesine karşın ekleme ve silme işlemlerinde daha yavaştır. Kayıtların sayısı capacity order la orantılıdır.
- ✓ **İkili Arama Ağacı (Binary Search Tree):** İkili arama ağacında bir düğüm en fazla iki tane çocuğa sahip olabilir ve alt/çocuk bağlantıları belirli bir sırada yapılır.
- ✓ **Kodlama Ağacı (Coding Tree):** Bir kümedeki karakterlere kod ataması için kurulan ağaç şeklindedir. Bu tür ağaçlarda kökten başlayıp yapraklara kadar olan yol üzerindeki bağlantı değerleri kodu verir.
- ✓ **Sözlük Ağacı (Dictionary Tree):** Bir sözlükte bulunan sözcüklerin tutulması için kurulan bir ağaç şeklindedir. Amaç arama işlemini en performanslı bir şekilde yapılması ve belleğin optimum kullanılmasıdır.
- ✓ **Kümeleme Ağacı (Heap Tree):** Bir çeşit sıralama ağacıdır. Çocuk düğümler her zaman aile düğümlerinden daha küçük değerlere sahip olur. [4]

### 3.2 Ağaca veri ekleme

Teorik olarak ağaca veri eklenmesi olayı köke göre çocuk sayılan elemanlarının indisinin kök indisine göre soldan sağa doğru artacak şekilde artımsal olarak artması prensibine dayanır. Buna göre kök indisi  $i$  değerine sahipse, köke göre en soldaki ilk çocuk  $i+1$  numaralı indisi gösterir. En genel ağaç formatına yapılan bu tanımlama ikili ağaç için oldukça basitleşmektedir. Buna göre ikili ağaçta köke göre çocuk sayılan verilerden soldaki veri küçük olanı tutacağı için eklenecek eleman  $i$  numaralı indisten küçük ise yeni indis gözü  $2*i$  numaralı indisi, büyükse sağdaki çocuğu göstereceği için  $2*i+1$  numaralı indisi gösterir. Bu yorum algoritmanın tasarımını kolaylaştırmaktadır. Veri ekleme konusu ödevimizin uygulama konusu olduğu için algoritması ve gerçekleştirilmiş C kodu bölüm 4'te detaylarıyla paylaşılmıştır.

### 3.3 Ağaçtan veri silme

Ağaçların en temel dezavantajı eklenecek yeni elemanların dengeli ağaç özelliğini bozması ve ağacın dengesizleşmesi ve ağaçlarda silme algoritmalarının zorluğudur. Dizi, yığın, kuyruk gibi veri tiplerinde silme işlemleri oldukça basittir. Buna karşın ağaçtan veri bu kadar kolaylıkla silinememektedir.

Silme işlemine ait psedo kod şu şekilde verilebilir;

```

while(silinecek düğümün ve ailesinin adresini bulana kadar) {
q <-silinecek düğümün,
qa<-ailesinin adresi;
if(silinmek istenen bulunamadı ise)
yapacak bir şey yok dön;
if(silinecek düğümüm iki alt çocuğu da varsa)
sol alt ağacın en büyük değerli düğümünü bul;
(veya denge bozulmuş ise sağ alt ağacın
enküçük değerli düğümünü bul)
bu düğümdeki bilgiyi silinmek istenen düğüme aktar;
bu aşamada en fazla bir çocuğu olan düğümü sil;
silinen düğümün işgal ettiği bellek alanını serbest bırak;
}

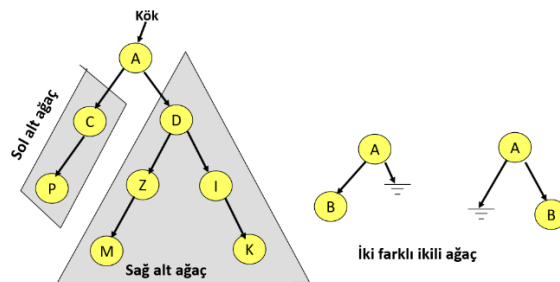
```

### 3.4 İkili arama ağacı

- ✓ Sonlu düğümler kümesidir. Bu küme boş bir küme olabilir (empty tree).

Boş değilse şu kurallara uyar.

- ✓ Kök olarak adlandırılan özel bir düğüm vardır.
- ✓ Her düğüm en fazla iki düğüme bağlıdır.
- ✓ Leftchild: Bir node'un sol işaretçisine bağlıdır.
- ✓ Right child: Bir node'un sağ işaretçisine bağlıdır.
- ✓ Kök hariç her düğüm bir daldan gelmektedir.
- ✓ Tüm düğümlerden yukarı doğru çıkıldıkça sonuçta köke ulaşılır.



Şekil-2 Örnek bir ikili ağaç

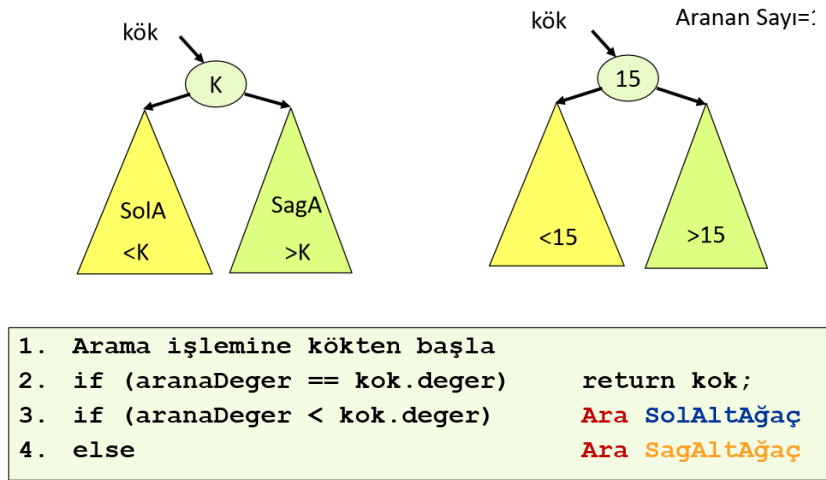
İkili arama ağacı ödevimizin konusu olması vesilesiyle 4. Bölümde kod ve algoritma olarak gerçekleştirilmiştir.

### 3.5 Ağaç üzerinde eleman arama

Bir k anahtarını aramak için, kök düğümünden başlayarak aşağı doğru bir yol izlenir.

Bir sonraki ziyaret edilecek düğüm, k anahtar değerinin geçerli düğümün anahtar değeriyle karşılaştırılması sonucuna bağlıdır.

Eğer yaprağa ulaşıldıysa anahtar bulunamamıştır ve null değer geri döndürülür. Şekil-3 de örnek bir arama algoritması şematize edilmiştir.



Şekil-3 Arama algoritması ve şematize edilmesi

Aynı algoritma while döngüsü yardımıyla yinelemeli şekilde yazılabilir. Yinelemeli versiyon özyinelemeli versiyona göre daha verimli çalışır.

### 3.6 Avantajları

- ✓ Aramada hız avantajları ağaçların en temel avantajlarını oluşturur.
- ✓ 2'nin katları şeklinde küçük ve büyük şeklinde yerleştirilmesi çok kısa adımda istenilen elemana ulaşılmasını sağlar. Ulaşılan bilgi kolayca güncellenebilir.
- ✓ Gerçeklenmesi çok karmaşık değildir. Zorluk silme ve dengeyi koruma noktalarında ortaya çıkmaktadır. Bunun için çok çeşitli algoritmalar ve tasarımlar bulunmaktadır.
- ✓ Bilgi gerçek dünyadaki var olan bir yapının bilgisayar dünyasına yansımaları olarak daha düzenli bir biçimde saklanır. Verilere ulaşım kolaylığı avantajı taşır.
- ✓ Uygun yerlerde kullanımı gereken bir veri yapısıdır.

## 4. Ödevin gerçekleştirilmesi

Bu bölümde algoritması derste işlenen ödev C programlama dilinde Dev C++ derleyicisinde gerçekleştirilmiş olup bölüm boyunca incelenmiş ve analiz edilmiştir.

### 4.1. Kodun yazılması

#### Kodun işlevleri:

- ✓ Kod ile dizi olarak tanımlanmış bir ağaç veri yapısı tasarlanmıştır. Bu veri yapısına eleman eklenebilmektedir.
- ✓ Kod ile dinamik memory tahsisi yapılarak istenilen boyutta bir dizi boyutu dinamik olarak ayarlanabilir
- ✓ Kod ile dizinin dolayısıyla ağacın bütün gözleri sıfırlanabilir
- ✓ Kod ile diziye rastgele değerli yerleştirilir. Bu veriler ağaç veri yapısına uygun yerleştirilir. Ve duruma şartı olarak dizi boyutu esas alınır.
- ✓ Kod ile ikili ağaçta arama yapılabilir
- ✓ Kod ile lineer olarak arama yapılabilir. Buradaki maksat ödev çıktılarından biri olan ağacın gücünün gözlemlenmesidir. Aynı şartlarda yapılan aramalar için adım sayısı ekrana yazdırılarak farklar çıktılar bölümünde değerlendirilmiştir.
- ✓ Kod ile ağaç veri yapısı bir dizi halinde yan yana oluşan elemanlar şeklinde gözlemlenebilir
- ✓ Kod her şeyden önce kullanıcı dostu olmayı hedeflemiş olup, Menü ile ve diğer çıktılarla kullanıcıyı yönlendirebilir.
- ✓ Kod da gereksiz ve yersiz döngülerden kaçınılarak daha sade bir algoritma kurulması hedeflenmiştir.

Yazılan C kodu aşağıda paylaşılmıştır.

```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#include <time.h>

int main(){
    srand((unsigned)time(0));
    int boyut=0,secim=1,i=1,aranan=1,eleman=1,rastgele, adimsayisi;
    printf("Agaciniza girilebilecek maksimum eleman sayisini giriniz:\n");
    scanf("%d",&boyut); //Maksimum eleman sayısı kullanıcıdan alınarak dinamik dizi için yer
    ayırılacak
    int* dizi; // Ağaç veri yapısı dizi üzerinde tutulacak
    dizi=(int *)malloc((boyut*sizeof(int))); //Dinamik olarak dizi oluşturuldu

    for(i=0;i<boyut;i++) //Dizinin dolayısıyla ağacın bütün gözleri 0 bilgisini tutuyor
        dizi[i]=0; //Dizinin dolayısıyla ağacın bütün gözleri 0 bilgisini tutuyor
```



```

while(secim!=0){
printf("Menu\n");
printf("1-Eleman Ekleme icin 1'e basin\n");
printf("2-Eklediginiz elemani aramak icin 2'ye basin\n");
printf("3-Agaci lineer bir dizi olarak gozlemlemek icin 3'e basin\n");
printf("4-Agacin tum elemanlarini sifirlamak ve tum elemanlari silmek icin 4'e basin\n");
printf("5-Random elemanlarla butun dizi boyutu kadar elemanla doldurulmus rastgele bir agac yaratmak icin 5'e basin\n");
printf("6-Aramayi lineer bir diziyim gibi yapmak icin 6'ya basin (Binary Tree Search ile adim sayisini karsilastirmak icin\n");
printf("7-Programi tamamen sonlaktirmek icin 0'a basin\n");

```

```

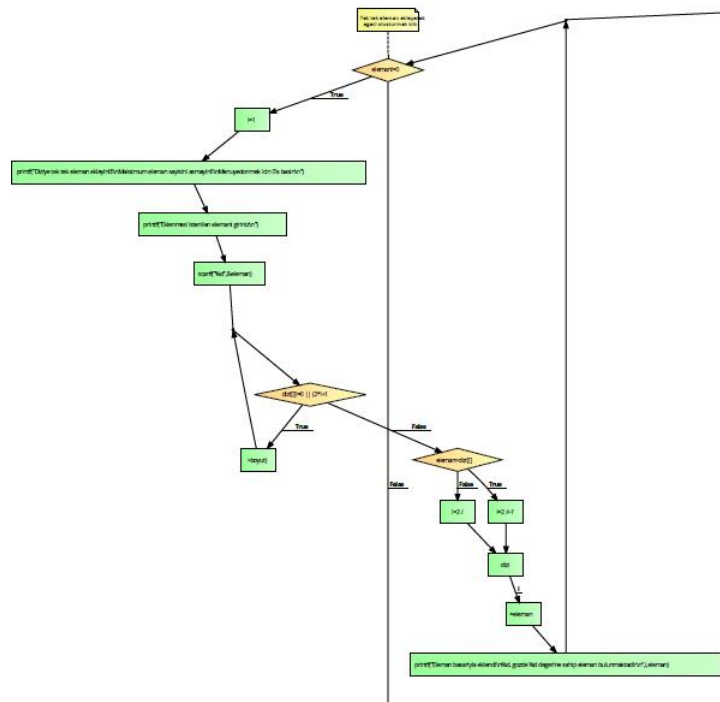
scanf("%d",&secim);
aranan=1;
eleman=1;

```

```

if(secim==1){          //Tek tek eleman ekleyerek agaci olusturmak icin
while(eleman!=0){
i=1;
printf("Diziye tek tek eleman ekleyin!!!\nMaksimum eleman sayisini asmayin!!\nMenuyedonmek icin 0'a basin\n");
printf("Eklennesi istenilen elemani giriniz\n");
scanf("%d",&eleman);
while(dizi[i]!=0 || (2*i+1)>boyut){
if(eleman>dizi[i])
i=2*i+1;
else
i=2*i;
}
dizi[i]=eleman;
printf("Eleman basariyla eklendi\n%d. gozde %d degerine sahip eleman bulunmaktadır\n",i,eleman);
}
}

```



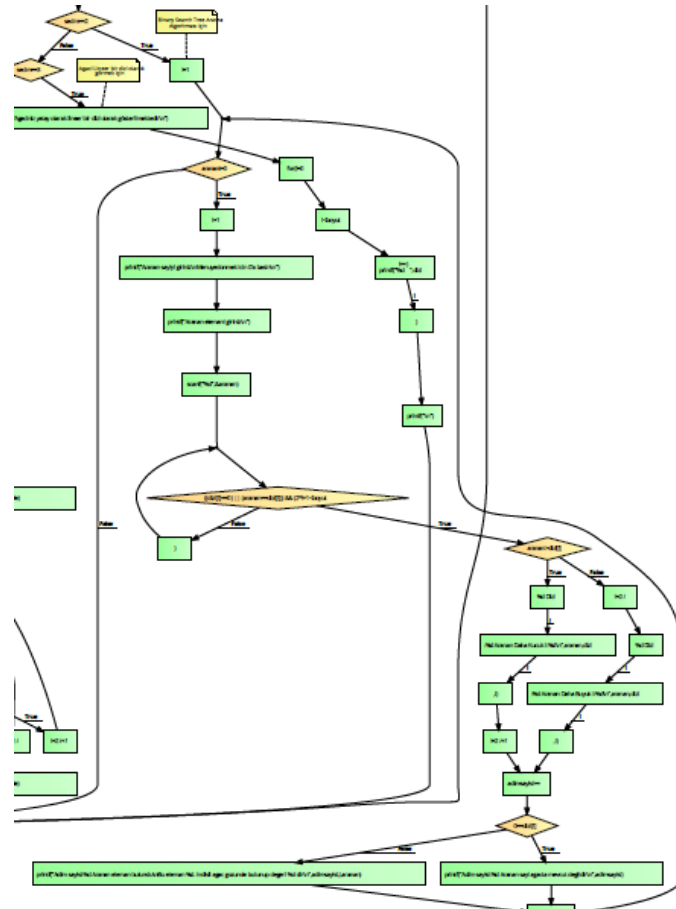
Şekil-4 İf(secim==1) koluna ait algoritmik akış (Tek tek eleman ekleme)

```

else if(secim==2){ //Binary Search Tree Arama Algoritması için
    i=1;
    while(aranan!=0){
        i=1;
        printf("Aranan sayiyi giriniz\nMenuyedonmek icin 0'a basin\n");
        printf("Aranan elemani giriniz\n");
        scanf("%d",&aranan);
        while(!((dizi[i]==0) || (aranan==dizi[i])) && (2*i+1<boyut)){
            if(aranan>dizi[i]){
                printf("Aranan:%d Dizi[i]:%d Aranan Daha Kucuk i:%d\n",aranan,dizi[i],i);
                i=2*i+1;
            }
            else{
                i=2*i;
                printf("Aranan:%d Dizi[i]:%d Aranan Daha Buyuk i:%d\n",aranan,dizi[i],i);
            }
            adimsayisi++;
        }
        if(0==dizi[i])
            printf("Adim sayisi:%d Aranan sayi agacta mevcut degildir\n",adimsayisi);
        else
            printf("Adim sayisi:%d Aranan eleman bulundu\nBu eleman %d. indisli agac gozunde bulunup degeri %d dir\n",adimsayisi,i,aranan);

        adimsayisi=0;
    }
}

```



Şekil-5 (if(secim==2) durumu için) Arama algoritması

```

else if (secim==3){ //Agaci Lineer bir dizi olarak görmek için
    printf("Agaciniz yatay olarak lineer bir dizi olarak gösterilmektedir\n");
    for(i=0;i<boyut;i++)
        printf("%d  ",dizi[i]);

    printf("\n");
}

else if(secim==4){ //Agacin tüm elemanlarını sıfırlamak için
    for(i=0;i<boyut;i++) //Dizinin dolayısıyla ağacın bütün gözleri 0 bilgisini tutuyor
        dizi[i]=0; //Dizinin dolayısıyla ağacın bütün gözleri 0 bilgisini tutuyor
}

else if (secim==5){ // Rastgele bir agac olusturmak icin, degerler agacin boyut sayisini asmayacagi ilk degerde
kesilir
    i=1;
    rastgele=rand()%60;
    dizi[1]=rastgele;
    while(2*i+1<boyut){

        rastgele=rand()%60;
        // while(dizi[i]!=0){
            if(rastgele>dizi[i])
                i=2*i+1;
            else
                i=2*i;
        //}
        dizi[i]=rastgele;
        printf("Rastgele eleman basariyla eklendi\n%d. gozde %d degerine sahip eleman bulunmaktadır\n",i,rastgele);
    }
}

else if (secim==6){ // Binary tree search ile karsilastirma yapabilmek icin adim sayisini gorebilecegimiz lineer
search algoritmasi
    i=0;

    printf("Lineer bir dizi gibi arama yapilacak\n");
    printf("Aranan sayiyi giriniz\nMenuyedonmek icin 0'a basin\n");
    printf("Aranan elemani giriniz\n");
    scanf("%d",&aranan);

    while (aranan!=dizi[i] && i<boyut){
        i++;
        adimsayisi++;
    }
    if(aranan==dizi[i])
        printf("Adim Sayisi:%d Aranan eleman bulundu\nBu eleman %d. indisli agac gozunde bulunup degeri %d
dir\n",adimsayisi,i,aranan);
    else
        printf("Adim Sayisi:%d Aranan sayi agacta mevcut degildir\n",adimsayisi);

    adimsayisi=0;
}

adimsayisi=0;
}

printf("Program sonlandirildi\n"); //Program kullanıcı istegiyle sonlandirildi
getch();
return 0;

```

## 4.2 Çıktıların Elde Edilmesi ve Algoritma Analizi

**Menüye ait ekran çıktısı:** Menü ile kullanıcı yönlendirilmeye çalışılır ve yine kullanıcıya bu kapsamda 6 seçenek sunulur.

```
Ağacınıza girilebilecek maksimum eleman sayısını giriniz:
1000
Menu
1-Eleman Ekleme için 1'e basın
2-Eklediğiniz elemanını aramak için 2'ye basın
3-Ağacı lineer bir dizi olarak gözden geçirmek için 3'e basın
4-Ağacın tüm elemanlarını sıfırlamak ve tüm elemanları silmek için 4'e basın
5-Random elemanlarla bütün dizi boyutu kadar elemanla doldurulmuş rastgele bir a
gac yaratmak için 5'e basın
6-Aramayı lineer bir diziyim gibi yapmak için 6'ya basın (Binary Tree Search il
e adım sayısını karşılaştırmak için
7-Programi tamamen sonlandırmak için 0'a basın
```

Şekil-6 Menüye ait ekran çıktısı

**Diziye tek tek eleman eklenmesine ait ekran çıktısı:** Burada elemanlar tek tek kullanıcıdan gelen verilere göre yerleştirilir. Yerleşim işlemi ikili arama ağacına uygun şekilde gerçekleştirilir.

```
1
Diziye tek tek eleman ekleyin!!!
Maksimum eleman sayısını asmayın!!
Menüyedonmek için 0'a basın
Eklenmesi istenilen elemanı giriniz
12
Eleman başarıyla eklendi
1. gözde 12 değerine sahip eleman bulunmaktadır
Diziye tek tek eleman ekleyin!!!
Maksimum eleman sayısını asmayın!!
Menüyedonmek için 0'a basın
Eklenmesi istenilen elemanı giriniz
8
Eleman başarıyla eklendi
2. gözde 8 değerine sahip eleman bulunmaktadır
Diziye tek tek eleman ekleyin!!!
Maksimum eleman sayısını asmayın!!
Menüyedonmek için 0'a basın
Eklenmesi istenilen elemanı giriniz
15
Eleman başarıyla eklendi
3. gözde 15 değerine sahip eleman bulunmaktadır
Diziye tek tek eleman ekleyin!!!
Maksimum eleman sayısını asmayın!!
Menüyedonmek için 0'a basın
```

Şekil-7 Tek tek eleman ekleme metoduyla ağacı oluşturma

**Rastgele eleman eklenmesine ait ekran çıktısı:** Kullanıcı elemanları tek tek girmek istemez ve rastgele elemanların ağaca 2li ağaç algoritmasına uygun şekilde yerleştirmesini isterse menü üzerinden ilgili yere girerek bunu gerçekleştirebilir. Bu fonksiyonun durma koşulu başta kullanıcıdan alınan ve dinamik olarak belirlenen dizi boyutudur. Bu boyut doğal olarak ağacın seviyesini de sınırlandırmaktadır.

```

1. gozde 33 degerine sahip eleman bulunmaktadır
Rastgele eleman basariyla eklendi
2. gozde 28 degerine sahip eleman bulunmaktadır

```

Şekil-8 Rastgele eleman ekleme fonksiyonu

**Ağacın yatay lineer bir dizi olarak gösterilmesi:** Kullanıcı ağaçta tutulan veriyi gözlemlemek için bu modülü kullanabilmektedir.

```

Ağaciniz yatay olarak lineer bir dizi olarak gösterilmektedir
0 33 24 46 18 27 0 54 7 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

```

Şekil- 9 Ağacın dizi olarak gösterilmesi

**Ağaçtaki bütün verilerin silinmesi:** Kullanıcı bütün verileri yok etmek ve aynı dizi üzerinde yeni ağaç verileri tanımlamak için bu modülü kullanabilir. Şekil- de görülen veriler bu modül kullanıldıktan sonra aşağıdaki gibi sıfırlanmıştır.

```

Ağaciniz yatay olarak lineer bir dizi olarak gösterilmektedir
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

```

Şekil-10 Ağacın tüm verilerinin silinmesi

**Verileri aranması:**

**BST Arama Algoritması:** Bu modül ile kullanıcı eklediği verileri veya random olarak yaratılmış ağaç verilerini arayarak bu veriler ile ilgili bilgilere ulaşabilmektedir.

```

Aranan:7 Dizilil:24 Aranan Daha Buyuk i:2
Aranan:7 Dizilil:18 Aranan Daha Buyuk i:4
Aranan:7 Dizilil:7 Aranan Daha Buyuk i:8
Adim sayisi:3 Aranan eleman bulundu
Bu eleman 8. indisli agac gozunde bulunup degeri 7 dir
Aranan sayiyi giriniz
Menuyedonmek icin 0'a basin
Aranan elemani giriniz

```

Şekil- 11 BST ile ağaç üzerinde arama

Yukarıdaki örnekte de görüldüğü gibi kod algoritma analizine yardımcı olmaktadır. Bu örnekte aranan sayı 3 adımda bulunmuştur. Lineer aramada bu adım sayısı 3 ile karşılaştırılmalıdır.

```

Aranan:42 Dizilil:33 Aranan Daha Kucuk i:1
Aranan:42 Dizilil:0 Aranan Daha Buyuk i:6
Adim sayisi:2 Aranan sayi agacta mevcut degildir
Aranan sayiyi giriniz
Menuyedonmek icin 0'a basin
Aranan elemani giriniz

```

Şekil- 12 BST ile ağaç üzerinde arama(2)

Yukarıdaki örnekte ise aranan sayı bulunamamıştır.

**Lineer Arama Algoritmasıyla:** Kullanıcı arama işlevi olarak lineer bir dizi gibi arama yapabilir. Bunun amacı BST ile Lineer arama algoritması karşılaştırmaktır. Bu örnekte ele alınan az sayıda veriye rağmen BST nin çok daha avantajlı olduğu gerçeği ortaya çıkmaktadır.

```
Lineer bir dizi gibi arama yapılacak
Aranan sayiyi giriniz
Menuyedonmek için 0'a basın
Aranan elemani giriniz
54
Adim Sayisi:7 Aranan eleman bulundu
Bu eleman 7. indisli agac gozunde bulunup degeri 54 dir
```

Şekil- 13 Lineer dizi üzerinde klasik arama algoritması

**Programın başarıyla sonlandırılması:** Kullanıcı menü ekranındayken 0 tuşuna basarsa program başarıyla sonlandırılır.

```
Program sonlandırildi
```

Şekil- 14 Programın sonlandırılması

#### 4.3 Sonuçlar ve Yorumlanması

Kodun uç değerlerde gözetilerek algoritma analizi gerçekleştirilmiş olup doğru çalıştığı gözlemlenmiştir. Ancak kullanıcı negatif değerler yâda int veri tipi dışında veri tipleri girmemelidir. Ancak bu girişleri için de ufak kod değişikliğiyle sistem yeni haline adapte edilebilir.

Sonuçta ödevle birlikte ağaç veri yapısına C kodu gerçekleştirilmiş olup, arama algoritması olarak üstünlüğü bizzat lineer arama ile karşılaştırılmış ve gözlemlenmiştir.

Ödev ile en basit veri yapısı olan diziler, dinamik olarak yer tahsisi, ağaçlar, ikili arama ağacı oluşturulması, sıralanması ve bütün verilerin silinmesine yönelik farklı senaryolar algoritmalaştırılarak kod haline dönüştürülmüştür.

Böylece ödevde ödevin giriş kısmında amaçlanan çıktılar gerçekleştirilerek, denenmiş ve kavranılmaya çalışılmıştır.

#### 5.Kaynakça

[1] [http://tr.wikipedia.org/wiki/Veri\\_yap%C4%B1s%C4%B1](http://tr.wikipedia.org/wiki/Veri_yap%C4%B1s%C4%B1) (İnternet kaynağı)

[2] <http://bilgisayarkavramlari.sadievrenseker.com/2007/05/03/linked-list-linkli-liste-veya-bagli-liste/> (İnternet kaynağı)

[3] <http://ceng.gazi.edu.tr/~akcayol/files/DSL6Trees.pdf> (İnternet kaynağı)

[4][http://firatyazilim.com/dosyalar/2.Sinif\\_\\_1.Donem/Veri\\_Yapilari/5.Hafta%20\\_Agaclar.pdf](http://firatyazilim.com/dosyalar/2.Sinif__1.Donem/Veri_Yapilari/5.Hafta%20_Agaclar.pdf) (İnternet Kaynağı)