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

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 gerçekleme 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 oluşturmayan graf veri yapısı olduğu için her ağaç verisi yapısı aynı zamanda bir graf veri yapısının özel bir halidir.

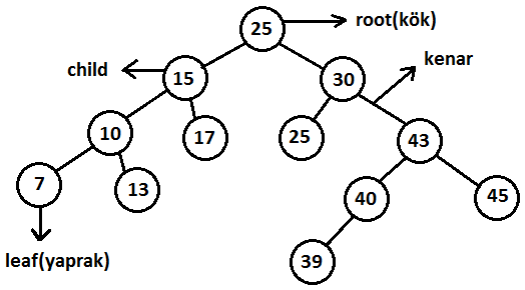
Sıklıkla kullanılan veri yapıları bu şekilde özetlenebilir. Böylece bu bölümde veri yapıları en basit tanımlarıyla incelenmiştir. Bölüm 3. te ödevimizin konusu olan ağaç veri yapısı ele alınmıştır.

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- de belirtilmiştir.



Ağaç yapısı

Şekil-

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 şeklidir. Bu tür ağaçlarda kökten başlayıp yapraklara kadar olan yol üzerindeki bağlantı değerleri kodu verir.
- ✓ **Sözlük Ağacı(Dictonary Tree):**Bir sözlükte bulunan sözcüklerin tutulması için kurulan bir ağaç şeklidir. 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 gerçeklenmiş C kodu bölüm 4'te detaylarıyla paylaşılmıştır.

3.3 Ağaçtan veri silme

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.

3.5 Ağaç üzerinde eleman arama

3.6 Avantajları

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.

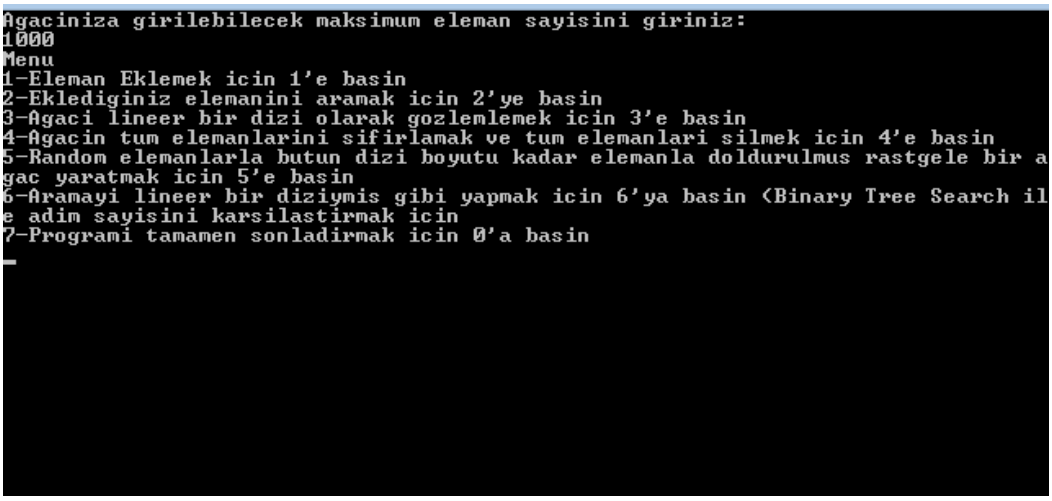
```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#include <time.h>

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 dolayısla ağacın bütün gözleri 0 bilgisini tutuyor
  dizi[i]=0;          //Dizinin dolayısla ağacın bütün gözleri 0 bilgisini tutuyor

  while(secim!=0){
  printf("Menu\n");
  printf("1-Eleman Eklemek icin 1'e basin\n");
  printf("2-Eklediginiz elemanini 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 diziymis gibi yapmak icin 6'ya basin (Binary Tree Search ile adim sayisini
karsilastirmak icin\n");
  printf("7-Programi tamamen sonladirmak 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("Eklenmesi 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 bulunmaktadir\n",i,eleman);
}
}


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;
  }
}

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 icin
             for(i=0;i<boyut;i++)     //Dizinin dolayısla ağacın bütün gözleri 0 bilgisini tutuyor
     dizi[i]=0;        //Dizinin dolayısla 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 bulunmaktadir\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 kullanici istegiyle sonlandirildi
getch();
return 0;
}
```

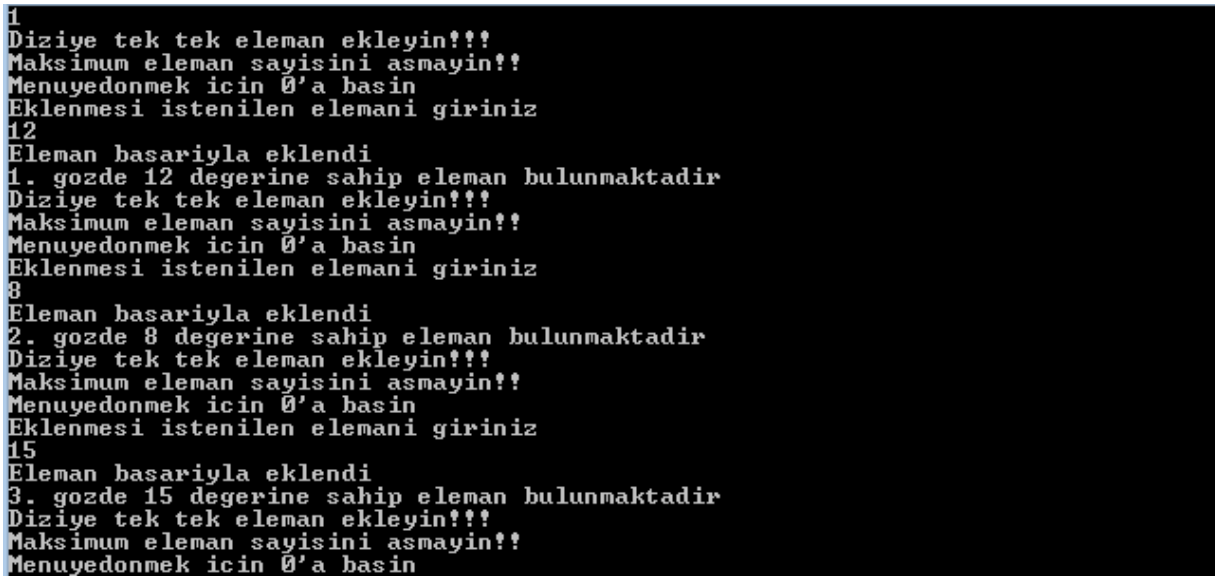
4.2 Çıktıların Elde Edilmesi ve Algoritma Analizi

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



Şekil- 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.

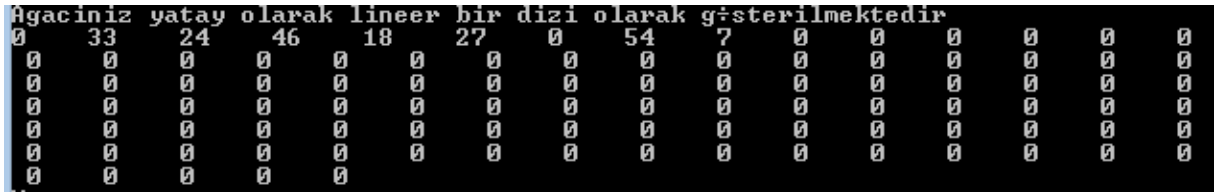


Şekil- Tek tek eleman ekleme metoduyla ağacı oluşturma

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.

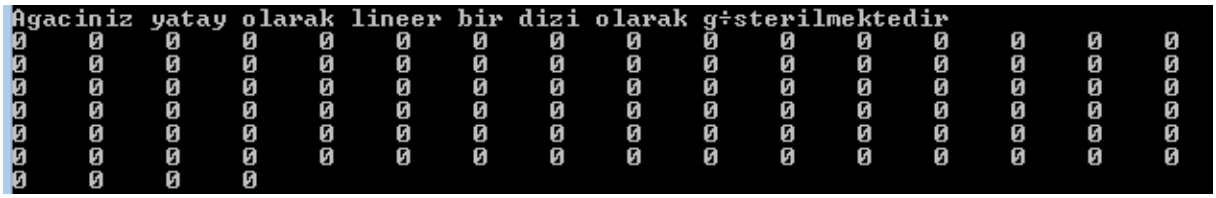
Şekil-

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



Şekil-

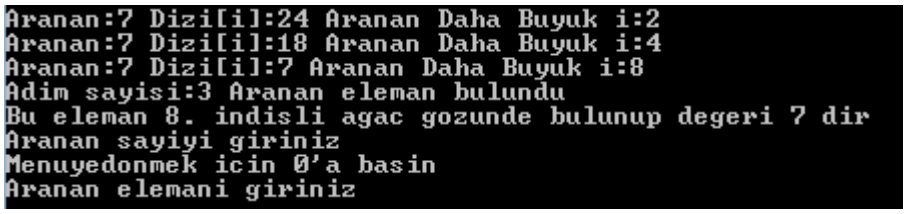
Ağaçtaki bütün verilerin silinmesi: Kullanıcı bütün verileri yok etmek ve aynı dizi üzerinde yeni ağaç verileri tanımlamak için bu modülü kullanabilir. Şekil- de görülen veriler bu modül kullanıldıktan sonra aşağıdaki gibi sıfırlanmıştır.



Şekil-

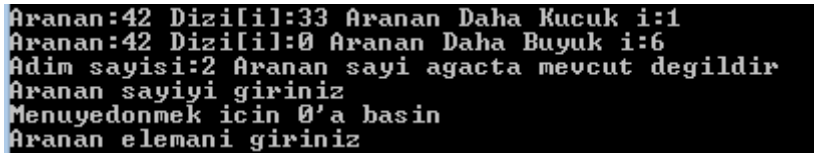
Verileri aranması:

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



Şekil-

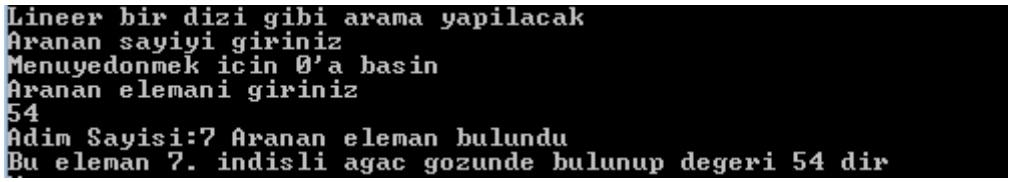
Yukarıdaki örnekte de görüldüğü gibi kod algoritma analizine yardımcı olmaktadır. Bu örnekte aranan sayı 3 adımda bulunmuştur. Lineer aramada bu adım sayısı 3 ile karşılaştırılmalıdır.



Şekil-

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.



Şekil-

Programın başarıyla 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.

5.Kaynakça

[1] 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 (İnternet Kaynağı)