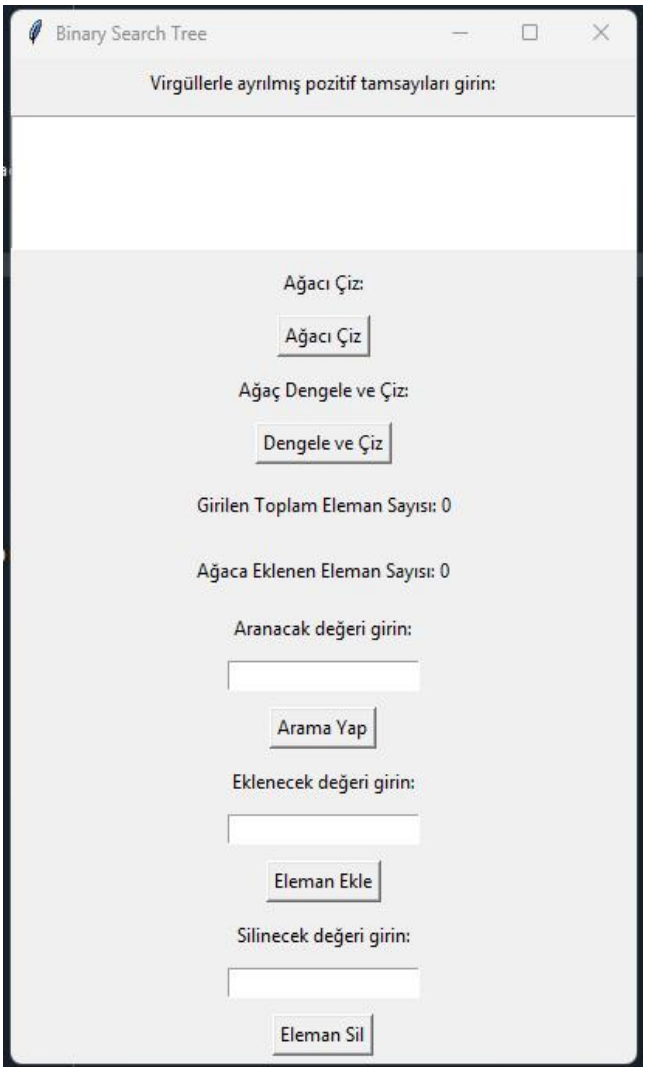
Yıldız Teknik Üniversitesi
Bilgi Teknolojileri TYL Programı
2024-2025 Güz YY
BLM5226 - Veri Yapıları ve Algoritma Tasarımı
Proje Raporu

Emirhan Eren-24574219

Ali Köprülü-24574244

Projemizin Arayüzü :



- En yukarıdaki TextBox'a veriler aralara "," koyularak girilir.
- Ağacı Çiz butonu ile girilen değerler bir Binary Search Tree olarak çizilir.
- Dengele ve Çiz butonuna tıklanıldığında girilen değerler dengelenir ve dengeli bir ağac çizilir.
- Girilen toplam eleman bölümünde kullancının girdiği toplam eleman sayısı tutulur, Ağaca eklenen eleman sayısı kısmında ise ağaca eklenebilen eleman sayısını gösterir.
- Arama TextBox'ına girilen değer alınır ve ağaç üzerinde aranır. Değer bulunursa o düğüm yeşil renk ile gösterilir.
- Eklenecek Değer Textbox'a girildiğinde eklenecek değer hem en yukarıdaki textbox'a eklenir hem de yeni eklenen eleman ile ağaç çizilir. Yeni eklenen eleman ağaç üzerinde sarı renk ile gösterilir.
- Silinecek eleman TextBox'a girilir bu eleman ağaç üzerinde aranır. Eleman ağacın içerisinde varsa bu eleman silme kurallarına uygun olarak ağaçtan silinir ve yeni ağaç çizilir.

Kullanılan Fonksiyonlar ve Çalışma Çıktıları :

Proje iki adet python dosyasından oluşuyor. Bunlar ***AgacMetotları.py*** ve ***Main.py*** dosyaları. Bunları ayırmamızın sebebi ise daha düzenli bir kod yapısı sağlamak ve kullanılan fonksiyonların kod karmaşası yaratmasından kaçınmak.

Main.py içerisindeki fonksiyonlarımız ve kullandığımız kütüphaneler :

```
1    import tkinter as tk
2    from tkinter import messagebox
3    from AgacMetotlari import BSTDiziOlustur, AgacCiz, IndexAra, DugumBoya,DugumSil,AgacDengele
4    from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg
5    import matplotlib.pyplot as plt
```

Tkinter kütüphanesi bir kullanıcı arayüzü (GUI) oluşturmamızı sağlar. Bu arayüze textboxlar ve butonlar oluşturarak kullanıcının programla daha kolay ve etkili şekilde iletişime geçmesi sağlanır.

AgacMetotları bizim ağaç oluştururken kullandığımız fonksiyonların olduğu .py dosyası. Buradan kullandığımız fonksiyonları ***Main.py*** içerisine import ederek kullanıyoruz. Bu fonksiyonlar butonlarla etkileşime girildiğinde çağırılan metotların ağaç dizisi üzerinde yaptıkları işlemlerde kullanılıyor.

Matplotlib ise ağaçları görselleştirmek için kullandığımız kütüphane.

```
211  # GUI penceresi
212  root = tk.Tk()
213  root.title("Binary Search Tree")
214
215  # kullanıcıdan dizi alınacak input
216  tk.Label(root, text="Virgüllerle ayrılmış pozitif tamsayıları girin:").pack(pady=5)
217  txt_inputDizi = tk.Text(root, height=5, width=50)
218  txt_inputDizi.pack(pady=5)
219  # çizim butonu
220  tk.Label(root, text="Ağacı Çiz:").pack(pady=5)
221  btn_ciz = tk.Button(root, text="Ağacı Çiz", command=Ciz)
222  btn_ciz.pack(pady=5)
223  # Ağacı dengele ve çiz
224  tk.Label(root, text="Ağaç Dengele ve Çiz:").pack(pady=5)
225  btn_dengele=tk.Button(root,text="Dengele ve Çiz",command=Dengele)
226  btn_dengele.pack(pady=5)
227
228  #kullanıcının girdiği toplam eleman sayısını yazdırma
229  lbl_inputSayisi = tk.Label(root, text="Girilen Toplam Eleman Sayısı: 0")
230  lbl_inputSayisi.pack(pady=10)
231  #ağaca eklenen toplam veri
232  lbl_eklenenSayisi = tk.Label(root, text="Ağaca Eklenen Eleman Sayısı: 0")
233  lbl_eklenenSayisi.pack(pady=10)
234
235  # aranacak değerin girildiği textbox
236  tk.Label(root, text="Aranacak değeri girin:").pack(pady=5)
237  txt_inputAranacak = tk.Entry(root)
238  txt_inputAranacak.pack(pady=5)
239  # arama butonu
240  btn_ara = tk.Button(root, text="Arama Yap", command=Ara)
241  btn_ara.pack(pady=5)
242
243  # eklenecek değerin girildiği textbox
244  tk.Label(root, text="Eklenecek değeri girin:").pack(pady=5)
245  txt_inputEklenecek = tk.Entry(root)
246  txt_inputEklenecek.pack(pady=5)
247  # ekleme butonu
248  btn_ekle = tk.Button(root, text="Eleman Ekle", command=Ekle)
249  btn_ekle.pack(pady=5)
250
251  # silinecek değerin girildiği textbox
252  tk.Label(root, text="Silinecek değeri girin:").pack(pady=5)
253  txt_inputSilinecek = tk.Entry(root)
254  txt_inputSilinecek.pack(pady=5)
255  # ekleme butonu
256  btn_sil = tk.Button(root, text="Eleman Sil", command=Sil)
257  btn_sil.pack(pady=5)
258
259  # GUI başlatma
260  root.mainloop()
261
```

Yukarıdaki görselde ise arayüzü oluşturuduğumuz komponentleri eklediğimiz kısım bulunmakta.

Başlangıçtan akış sırasıyla:

1. İlk olarak bir GUI penceresi oluşturuyoruz. root.title ile penceremizin başlığını set ediyoruz.

2. Ağaç oluşturulacak dizinin giriş yapılacağı TextBox'ı oluşturuyoruz. Bu TextBox üzerine kullanıcya bilgi vermek için bir label alanı ekliyoruz.

3. Bir label alanı ekleyip altına verilen inputlarla direkt olarak bir BST oluşturmak için buton ekliyoruz. Bu buton tıklanıldığında Ciz() fonksiyonunu çağırıyor. Bu fonksiyon giriş verilerini bir Ağaç yapısına dönüştürüp ekrana bir ağaç çıktısı veriyor.

4. Bir label alanı ekleyip altına verilen inputlarla Dengeli BST oluşturmak için buton ekliyoruz. Bu butona tıklanıldığında Dengele() fonksiyonu çağırılıyor. Bu fonksiyon verilen verileri Dengeli BST oluşturmak için işliyor ve ekrana bir Dengeli BST çıktısı veriyor.

5. İki adet label ekleniyor. İlk label kullanıcının girdiği toplam eleman sayısı GUI üzerinde gösteriliyor. RAM kısıtından dolayı kullanıcının belli bir adetten fazla girdiği veriler için uygun yer ayıramadığımız durumlarda burada ağaca eklenen eleman sayısını gösteriyoruz.

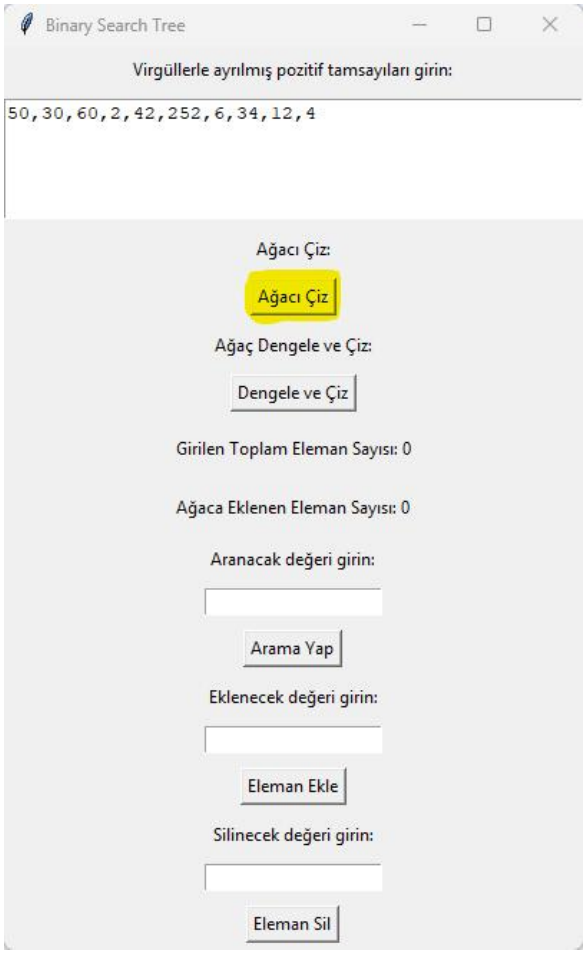
6. Arancak değerin girilmesi için GUI üzerine bir TextBox ekleniyor. Altına bir arama butonu ekleniyor ve buton tıklanıldığı zaman Ara() fonksiyonunu çağırıyor. Bu fonksiyon ağaçta elemanı arıyor ve bulunursa bulunduğu düğümü yeşil renge boyayarak kullanıcıya bu ağacı çiziyor.

7. Eklenecek değerin girilmesi için GUI üzerine bir TextBox ekleniyor. Altına da bu değerin ağaca eklenmesi için bir buton ekleniyor. Butona basıldığında buradaki değeri hem kullanıcının ilk giriş yaptığı TextBox'a hem de ağaca ekleyerek GUI üzerine bu ağacı çiziyor. Butona tıklanıldığı zaman Ekle() fonksiyonu çalışıyor.

8. Silinecek değerin girilmesi için GUI üzerine bir TextBox ekleniyor. Altına da silme işleminin ağaç üzerinde gerçekleştirilmesi için bir buton ekleniyor. Bu butona tıklanıldığı zaman hem girilen eleman ağaç üzerinden, hem de kullanıcının ilk giriş yaptığı TextBox üzerinden bu değer siliniyor ve silindikten sonra GUI üzerinde ağaç çiziliyor.

Eklenen label,TextBox,Buton gibi komponentler ilk adımda oluşturulan root penceresine bağlanıyor. Her bir komponent arasına pady ile 5 birimlik boşluk bırakılıyor ve bu sayede GUI üzerinde tasarım sağlanıyor.

Ağaç Oluşturma için Kullanılan Fonksiyonlar :



İlk olarak kullanıcı giriş değerlerini “,” ile ayırarak girer ve Ağacı Çiz butonuna tıklar. Bu buton Ciz() fonksiyonunu tetikler.

```python
def Ciz():
    # textboxtan değerleri al
    inputData = txt_inputDizi.get("1.0", tk.END).strip()
    global numbers #numbers'ı global yapmamız gerekiyor çünkü Ara fonksiyonunda bu listeye ihtiyacımız olacak
    try:
        # textboxtan alınan değerleri , ile ayırarak bir listeye at
        numbers = [int(num.strip()) for num in inputData.split(",") if num.strip()]
    except ValueError:
        messagebox.showerror("Hata", "Lütfen yalnızca sayılardan oluşan bir liste girin!")
        return

    if len(numbers) > 100:
        messagebox.showerror("Hata", "Maksimum 100 elemanlı bir liste girebilirsiniz!")
        return

    # 100 elemanlı bir dizi başlat (hepsi 0)
    AgacDizisi = [0] *(2**25)   #4000 #((2**len(numbers)-1))

    # aldığımız input listesini BST dizisine dönüştür
    for number in numbers:
        AgacDizisi = BSTDiziOlustur(AgacDizisi, number)


    # BST çizimini oluştur
    fig = AgacCiz(AgacDizisi)

    new_window = tk.Toplevel(root)
    new_window.title("Binary Search Tree")

    # Matplotlib grafiğini tkinter penceresine yerleştir
    canvas = FigureCanvasTkAgg(fig, master=new_window)
    canvas_widget = canvas.get_tk_widget()
    canvas_widget.pack()
    canvas.draw()

    plt.close('all')

    # kullanıcının girdiği eleman sayısını gui üzerine yazdırma
    girilenElemanSayisi = len(numbers)
    lbl_inputSayisi.config(text=f"Girilen Toplam Eleman Sayısı : {girilenElemanSayisi}")

    #agaç dizisi için gerekli yer ayrılamadığında ağaca eklenen toplam eleman sayısını gui üzerine yazdırır
    eklenenElemanSayisi=0
    for i in range(len(AgacDizisi)):
        if(AgacDizisi[i]!=0):
            eklenenElemanSayisi=eklenenElemanSayisi+1
    lbl_eklenenSayisi.config(text=f"Ağaca Eklenen Eleman Sayısı : {eklenenElemanSayisi}")
```

Daha sonrasında bu fonksiyon çalışmaya başlar. Burada sırasıyla :

İlk olarak kullanıcının girdiği txt_inputDizi TextBox'ından değerler okunur.

Sonrasında global bir numbers değişkeni oluşturulur. Bu değişkenin global olmasının nedeni diğer fonksiyonlar üzerinden de bu değişkene ulaşıp bu dizi üzerinde işlemler yapmak istememiz.

Sonrasında bir try-catch bloğunda kullanıcının girdiği değerleri "," ile ayırarak başlıyoruz. Burada kullanıcının doğru formatta mı giriş yaptığını ve girilen sayıların 100'den fazla olup olmadığını da kontrol ediyoruz. Eğer yanlış giriş yaptıysa bir MessageBox ile GUI'de hatayı gösteriyoruz.

Sonrasında bir AgacDizisi oluşturuyoruz. Bu dizinin boyutu tamamen dengesiz bir ağaç olduğu varsayılırsa eğer 2^(100)-1 olmalı fakat bizim RAM'imizde o kadar yer olmadığı için programın çalışabileceği bir değer girerek ağaç dizimizi oluşturuyoruz.

Sonrasında numbers dizisinindeki elemanları tek tek dolaşarak BSTDiziOlustur fonksiyonunuçağırıyoruz. BSTDiziOlustur fonksiyonu ***AgacMetotları.py*** içerisinden çağırılıyor.

```python
def BSTDiziOlustur(AgacDizisi, inputValue):
    # textbox'tan okunan değerleri BTS yapısına uygun bir diziye aktarır
    if(AgacDizisi[0]==0):#Kok değeri 0 ise gelen değeri başlangıç kök değeri yapar
        AgacDizisi[0]=inputValue
    else:
        kokIndeks = 0
        while kokIndeks < len(AgacDizisi):
            # sol alt ağaca mı ?
            if inputValue < AgacDizisi[kokIndeks]:
                solIndeks = 2 * kokIndeks + 1  # solun indeksini al
                if solIndeks <= len(AgacDizisi) and AgacDizisi[solIndeks] == 0:  # solunda yer var mı ?
                    AgacDizisi[solIndeks] = inputValue #solunda yer varsa yerleş
                    break
                kokIndeks = solIndeks  # yer yoksa solu kök olarak al
                # sağ alt ağaç mı ?
            elif inputValue > AgacDizisi[kokIndeks]:
                    sagIndeks = 2 * kokIndeks + 2  # sağın indeksini al
                    if sagIndeks <= len(AgacDizisi) and AgacDizisi[sagIndeks] == 0:  # sağında yer var mı ?
                        AgacDizisi[sagIndeks] = inputValue
                        break
                    kokIndeks = sagIndeks  # sağında yer varsa yerleş
            else:
                break  #aynı değer varsa ekleme
    return AgacDizisi
```

Bu fonksiyon parametre olarak biz Dizi ve eklenecek değeri alıyor. Burada ikili arama ağacı kurallarına uyularak ikili arama ağacı oluştuluyor. Parametre olarak verdiğimiz dizide boş olan değerler 0 ile ifade ediliyor. İlk gelen değer için eğe kök değer 0 ise ilk gelen değeri kök değeri yapıyor. Kök değer var ise eklenecek olan değeri kök ile karşılaştırıyor ve buradaki karşılaştırma sonucuna göre kökün sol düğümüne veya sağ düğümüne yönlenerek karşılaştırmalarına devam ediyor.

Karşılaştırmalar sonucunda uygun olarak bulunan yani 0 olan düğüme bu değer yerleştiriliyor. Arama işlemi indeksler AgacDizisi'nin boyutunu geçmediği sürece yapılıyor.

Değer geliyor, köke bakılıyor, kök dolu ise kökle değer karşılaştırılıyor,küçükse sol alt indisi kök kabul ediliyor, büyük ise sağ alt indisi kök kabul ediyor sonrasında bu döngüyü tamamlayana kadar AgacDizisi içerisinde geziyor ve AgacDizi'ni oluşturup ***Main.py*** içerisinde çağırıldığı yere dönüyor.

AgacDizisi oluşturulduktan sonra AgacCiz() fonksiyonu çağırılıyor. Bu fonksiyon ***AgacMetotları.py*** içerisinde yer alıyor.

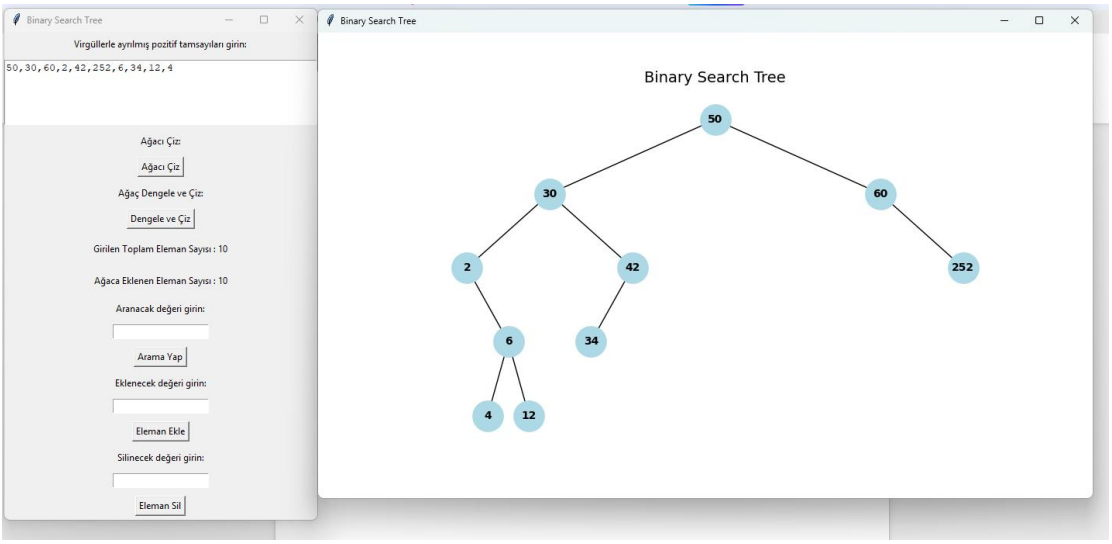
```
def AgacCiz(AgacDizisi):
    #ağaç dizisi ile ağacı görsel olarak çizer
    graph = nx.DiGraph()
    pos = {}
    pos = AgacGraphOlustur(graph, AgacDizisi, pos)

    # graph ayarlarını ekleme
    fig, ax = plt.subplots(figsize=(10, 6))
    nx.draw(
        graph, pos, with_labels=True, node_size=800, node_color="lightblue",
        font_size=10, font_weight="bold", arrows=False, ax=ax
    )
    ax.set_title("Binary Search Tree", fontsize=14)

    #eksenleri kapama
    ax.axis("off")
    return fig
```

Burada network kütüphanesi kullanılarak ağaç dizisi içerisindeki değerleri birer birer bağlanıyor. Bu bağlama işlemi AgacGraphOlustur fonksiyonu içerisinde gerçekleşiyor.

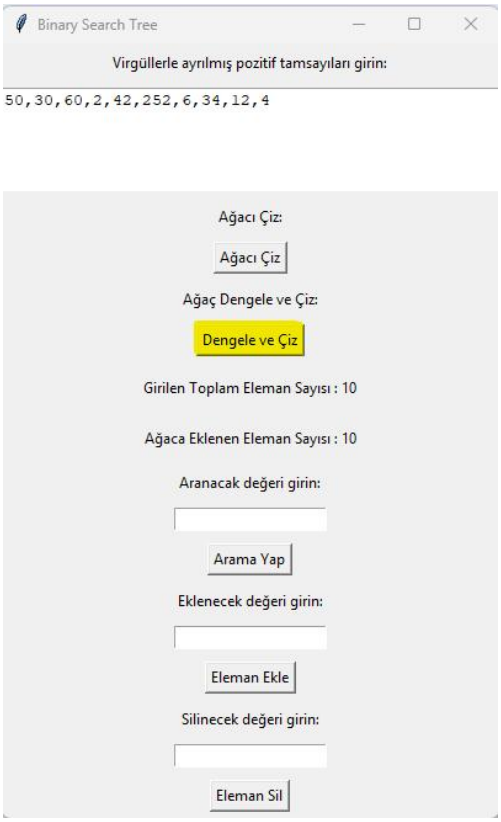
```
def AgacGraphOlustur(graph, AgacDizisi, pos=None, index=0, x=0, y=0, layer=1):
    #ağaç dizisini kullanarak düğümler arasında kenarları oluşturur
    if index >= len(AgacDizisi) or AgacDizisi[index] == 0:
        return pos

    pos = pos or {}
    pos[AgacDizisi[index]] = (x, y)

    # sağ-sol indekslerin hesaplamaları
    left_index = 2 * index + 1
    right_index = 2 * index + 2

    #kendini recursice olarak çağırarak agaç dizisi içerisinde sol ve sağ düğümleri arayarak birer kenar çizer
    if left_index < len(AgacDizisi) and AgacDizisi[left_index] != 0:
        graph.add_edge(AgacDizisi[index], AgacDizisi[left_index])
        AgacGraphOlustur(graph, AgacDizisi, pos, left_index, x - 1 / (2 ** layer), y - 1, layer + 1)
    if right_index < len(AgacDizisi) and AgacDizisi[right_index] != 0:
        graph.add_edge(AgacDizisi[index], AgacDizisi[right_index])
        AgacGraphOlustur(graph, AgacDizisi, pos, right_index, x + 1 / (2 ** layer), y - 1, layer + 1)

    return pos
```

Bu fonksiyonda ağaç içerisini tek tek gezerek bir graph oluşturuyor. Bu graph ta her değeri bir düğüm ve bağlandığı düğümü de yine sol çocuk, sağ çocuk indis hesabı ile yapıyor. Düğümler arasına gerekli bağlantılar çiziliyor. Bu graph yapısı AgacCiz() fonksiyonu içerisine dönülüyor.

Daha sonrasında elimizde bu ağaca ait bir graph yapısı oluyor. Bu graph yapısı ise Ciz() metodu içerisinde yeni bir pencere açılarak kullanıcıya çiziliyor.



Dengeli ağaç oluşturmak için yine aynı çizim fonksiyonları kullanılıyor. Dengeli ağaç çizmek için kullanıcı “,” ile değerleri girer ve Dengele Çiz butonuna basar.



Bu buton Dengele() fonksiyonunu çağırır.

```
def Dengele():
    inputData = txt_inputDizi.get("1.0", tk.END).strip()
    global numbers #numbers'ı global yapmamız gerekiyor çünkü Ara fonksiyonunda bu listeye ihtiyacımız olacak
    try:
        # textboxtan alınan değerleri , ile ayırarak bir listeye at
        numbers = [int(num.strip()) for num in inputData.split(",") if num.strip()]
    except ValueError:
        messagebox.showerror("Hata", "Lütfen yalnızca sayılardan oluşan bir liste girin!")
        return

    if len(numbers) > 100:
        messagebox.showerror("Hata", "Maksimum 100 elemanlı bir liste girebilirsiniz!")
        return

    numbers = AgacDengele(numbers)
    print(numbers)
    # 100 elemanlı bir dizi başlat (hepsi 0)
    AgacDizisi = [0] *(2**25)   #4000 #((2**len(numbers)-1))

    # aldığımız input listesini BST dizisine dönüştür
    for number in numbers:
        AgacDizisi = BSTDiziOlustur(AgacDizisi, number)

    fig = AgacCiz(AgacDizisi)

    # kullanıcının girdiği eleman sayısını gui üzerine yazdırma
    girilenElemanSayisi = len(numbers)
    lbl_inputSayisi.config(text=f"Girilen Toplam Eleman Sayısı : {girilenElemanSayisi}")

    #agaç dizisi için gerekli yer ayrılamadığında ağaca eklenen toplam eleman sayısını gui üzerine yazdırır
    eklenenElemanSayisi=0
    for i in range(len(AgacDizisi)):
        if(AgacDizisi[i]!=0):
            eklenenElemanSayisi=eklenenElemanSayisi+1
    lbl_eklenenSayisi.config(text=f"Ağaca Eklenen Eleman Sayısı : {eklenenElemanSayisi}")

    new_window = tk.Toplevel(root)
    new_window.title("Dengelenmiş Ağaç")

    canvas = FigureCanvasTkAgg(fig, master=new_window)
    canvas_widget = canvas.get_tk_widget()
    canvas_widget.pack()
    canvas.draw()
    plt.close('all')
```

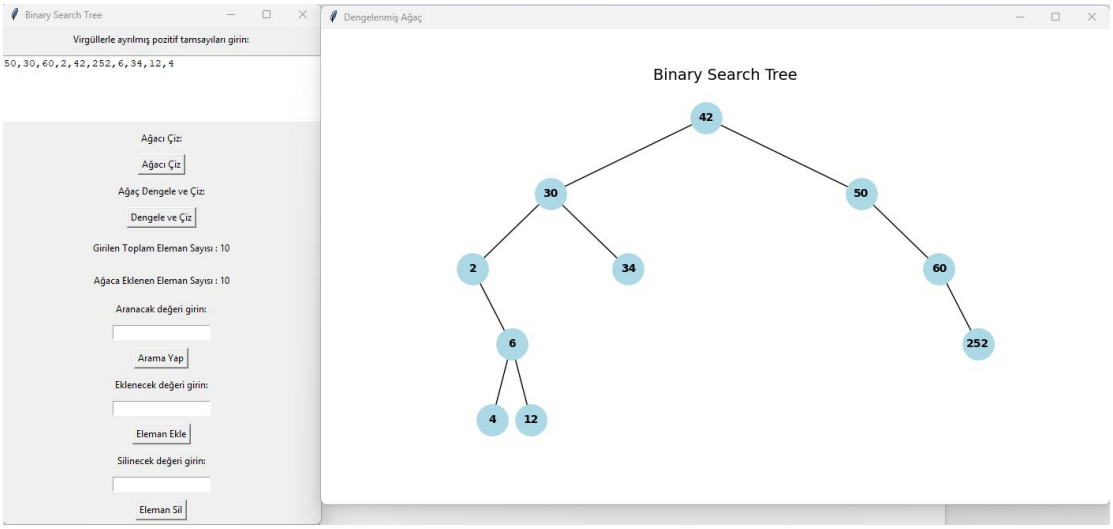
Buradaki fark kullanıcının girdiği inputlar numbers dizisine atandıktan sonra bu değerler dengeli bir ağaç oluşturmak için AgacDengele() fonksiyonunu kullanır. Bu fonksiyon **AgacMetotları.py** içerisinden çağırılır.

```
182     def BSTDengeliAgacOlustur(nums, start, end, tree):
183         if start > end:
184             return -1  #başlangıç bitiş kontrolü
185
186         mid = (start + end) // 2  # dizinin ortasındaki eleman alınır
187         root_index = len(tree)  # kök indisi için dizinin uzunluğu alınır
188         tree.append(nums[mid])  # Diziye kök elemanı ekle
189
190         # dizinin solu ve sağı için fonksiyon tekrarlı olarak çağırılır
191         BSTDengeliAgacOlustur(nums, start, mid - 1, tree)
192         BSTDengeliAgacOlustur(nums, mid + 1, end, tree)
193
194         return root_index  # kök indis
195
196     def AgacDengele(nums):
197         DengeliDizi = []  # dengeli numbers dizisi
198         BSTDengeliAgacOlustur(nums, 0, len(nums) - 1, DengeliDizi)  # diziyi oluştur
199         return DengeliDizi  # numbersı dön
```

AgacDengele fonksiyonu çağırıldıktan sonra parametre olarak kullanıcının girdiği numbers dizisini alır. Daha sonra kendi içerisinde BSTDengeliAgacOlustur() fonksiyonunu çağırır. Bu fonksiyon parametre olarak bir dizi, başlangıç değeri ve dönülecek olan diziyi alır. Bu fonksiyon dizinin ortasındaki değeri alır, bunu kök yapar ve sağında ve solunda kalan değerler için kendi kendini tekrardan çağırır.

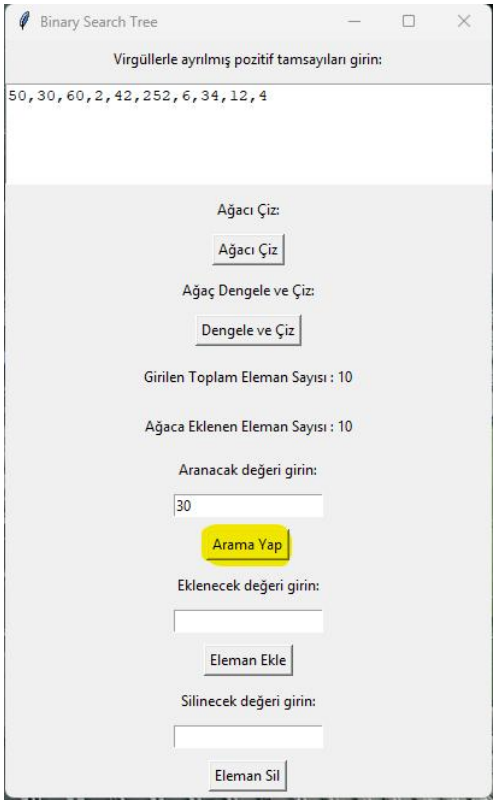
Ortanca değeri alır kök yapar ve solunda sağında kalan değerler için bunu tekrarlar, sol değerlerin ortanca değerini alır diziye ekler, sağ değerlerin ortanca değerini alır diziye ekler. Bu işlemi tekrarlar ve sonuç olarak bize Dengeli bir BST oluşturuabilmek için bir dizi döner.

Dengele() fonksiyonu içerisinde bu diziyi dolaşark AgacDizisi oluştururuz ve bu değeri AgacCiz() fonksiyonu ile kullanıcıya çizeriz.



Arama Fonksiyonu :

Kullanıcı Ağacını çizdikten sonra ağaç üzerinde değer arayabilir. Arayacağı değeri textbox'a girer ve arama butonuna tıklar. Bu buton Ara() fonksiyonunu tetikler.



```python
def Ara():
    #kullanıcıdan alınan değeri ağaçta arar ve eğer o değer varsa o düğümü yeşil renge boyar
    try:
        arananDeger = int(txt_inputAranacak.get())
    except ValueError:
        messagebox.showerror("Hata", "Lütfen geçerli bir sayı girin!")
        return

    # başlangıçtaki ağaç dizisi
    AgacDizisi = [0] *(2**25)

    # aldığımız input listesini BST dizisine dönüştür
    for number in numbers:
        AgacDizisi = BSTDiziOlustur(AgacDizisi, number)

    # input olarak alınan değerin indexini ara
    bulunanIndex = IndexAra(AgacDizisi, arananDeger)

    new_window = tk.Toplevel(root)
    new_window.title("Arama Sonucu")

    if bulunanIndex != -1:
        fig = DugumBoya(AgacDizisi, bulunanIndex,"green")
        canvas = FigureCanvasTkAgg(fig, master=new_window)
        canvas_widget = canvas.get_tk_widget()
        canvas_widget.pack()
        canvas.draw()
        plt.close('all')
    else:
        messagebox.showinfo("Hata", "Aranan değer bulunamadı !")
```

Ara fonksiyonu TextBox üzerinden değeri alır ve bunun int bir değer olup olmadığını kontrol eder. Eğer int değer değilse ekrana bir hata penceresi açar.

Kontrolden sonra bir AgacDizisi olusturur. Kullanıcının girdiği değerleri numbers dizisinden okuyarak BSTDiziOlustur ile bir BinarySearchTree oluşturur.

Daha sonra IndexAra() fonskiyonunu **AgacMetotları.py** içerisinden çağırır.

```python
def IndexAra(AgacDizisi, arananDeger):
    #ağaç dizisi üzerinde kullancının girdiği değeri arıyoruz
    # ilk kökten başlayarak arıyoruz
    idx = 0
    while idx < len(AgacDizisi):
        if AgacDizisi[idx] == arananDeger:
            return idx  #bulunan değerin indexi
        elif arananDeger < AgacDizisi[idx]:  # Sol alt ağaç
            idx = 2 * idx + 1
        elif arananDeger > AgacDizisi[idx]:  # Sağ alt ağaç
            idx = 2 * idx + 2
    return -1  # değer yoksa -1 dönülür
```

Bu fonksiyon AgacDizisi elemanlarını tek tek gezer. Parametre olarak AgacDizisi ve arananDeger değişkenlerini alır. Düğüme gider ve girilen değerin küçük olup olmadığını kontrol eder. Eğer küçükse sol büyükse sağ düğüme gider. Buradaki gitme işlemini BST indis hesabına göre yapar. Bu arama süreci indisin değerinin ağaç dizisinin boyutundan küçük olduğu sürece devam eder. Eğer değeri bulursa düğümün indisini döner, bulamazsa -1 döner. Bu bize değerin ağaçta olup olmadığını kontrol etmemiz için bir yol sunar.

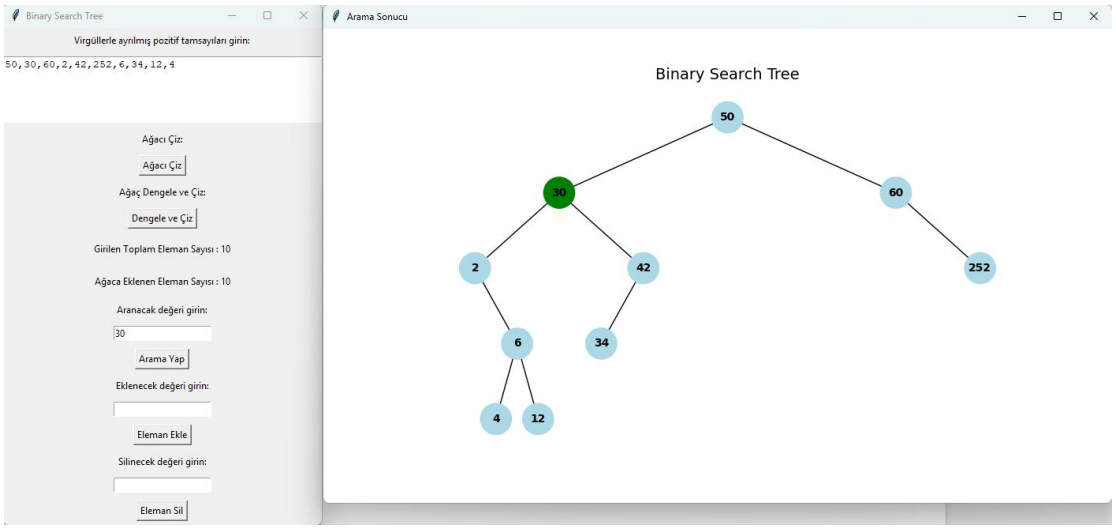
Daha sonra IndexAra() fonksiyonu Ara() fonksiyonuna aranan değer bulunduysa bu değerin indisini, bulunamadıysa -1 değerini döner. Bu dönüş değerine göre bir karşılaştırma yapılır.

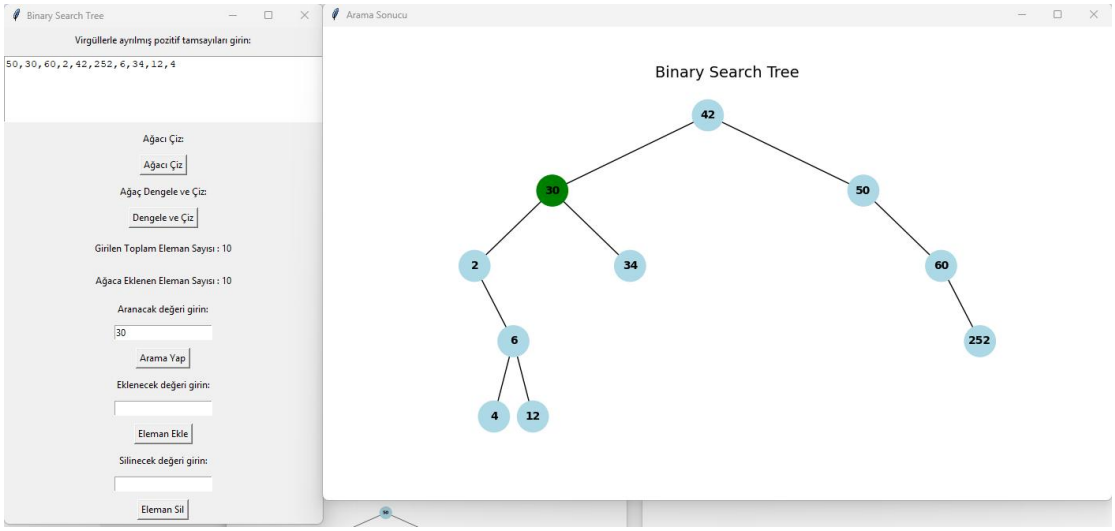
Eğer değer -1 değilse yani değer ağaçta bulunduysa DugumBoya() fonksiyonu çağırılır. Bu fonksiyon **AgacMetotları.py** içerisinden çağırılır.

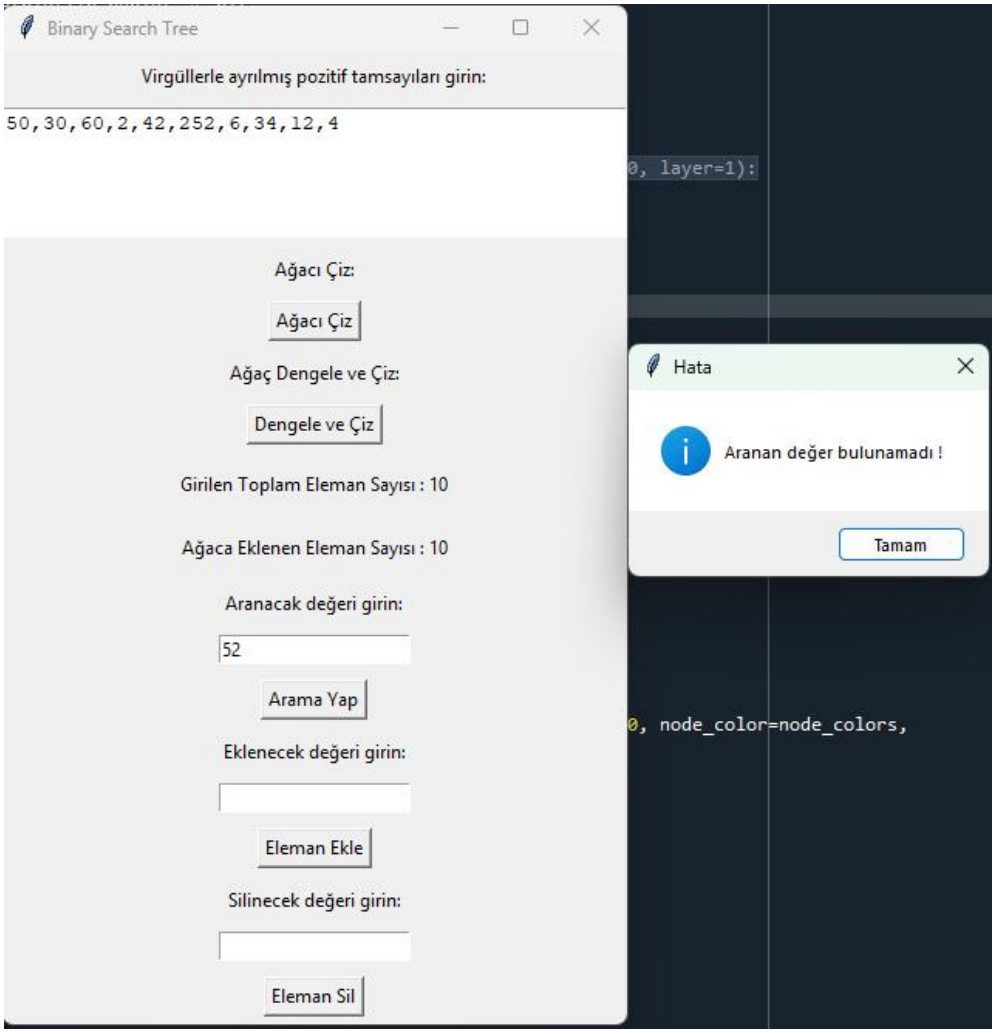
```python
def DugumBoya(AgacDizisi, boyanacakDugum,renk):
    graph = nx.DiGraph()
    pos = AgacGraphOlustur(graph, AgacDizisi)

    # Sıfır olmayan değerlerden düğüm listesi oluştur
    nodes = [value for value in AgacDizisi if value != 0]

    # Düğüm renklerini ayarla
    node_colors = []
    for idx, value in enumerate(AgacDizisi):
        if idx == boyanacakDugum:  # Highlight edilen indeks kontrolü
            node_colors.append(str(renk))
        elif value != 0:
            node_colors.append("lightblue")

    # Görselleştirme
    fig, ax = plt.subplots(figsize=(10, 6))
    nx.draw(
        graph, pos, with_labels=True, nodelist=nodes, node_size=800, node_color=node_colors,
        font_size=10, font_weight="bold", arrows=False, ax=ax
    )

    ax.set_title("Binary Search Tree", fontsize=14)
    ax.axis("off")  # Eksenleri kapat
    return fig
```

Burada her bir değer için bir node oluşturulur. Bu node'ların birer renk değeri vardır. Ağaç içeridine her bir düğüm tek tek kontrol edilir. Yolladığımız indisin renk değerini yeşil olarak işaretler, diğer düğümleri mavi şekilde temsil eder. Daha sonrasında bu çizimi Ara() fonksiyonu içerisine döner.

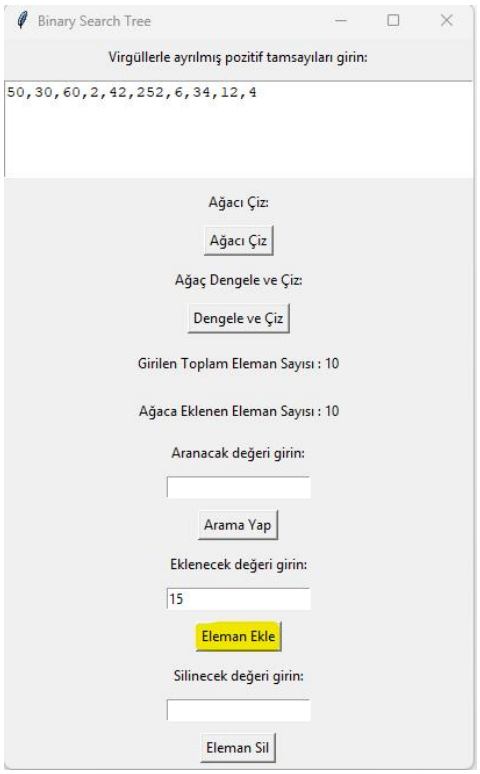
Ara() fonksiyonunda bu çizim kullanıcıya yeni bir pencere açılarak çizilir. Eğer değer ağaçta yoksa bir hata mesajı penceresi açılır.




Binary Search Tree
Virgüllerle ayrılmış pozitif tamsayıları girin:
50,30,60,2,42,252,6,34,12,4
Ağacı Çiz:
Ağacı Çiz
Ağaç Dengele ve Çiz:
Dengele ve Çiz
Girilen Toplam Eleman Sayısı : 10
Ağaca Eklenen Eleman Sayısı : 10
Aranacak değeri girin:
30
Arama Yap
Eklenecek değeri girin:
Eleman Ekle
Silinecek değeri girin:
Eleman Sil
Arama Sonucu
Binary Search Tree
42
30
50
2
34
60
6
252
4
12


Binary Search Tree
Virgüllerle ayrılmış pozitif tamsayıları girin:
50,30,60,2,42,252,6,34,12,4
Ağacı Çiz:
Ağacı Çiz
Ağaç Dengele ve Çiz:
Dengele ve Çiz
Girilen Toplam Eleman Sayısı : 10
Ağaca Eklenen Eleman Sayısı : 10
Aranacak değeri girin:
52
Arama Yap
Eklenecek değeri girin:
Eleman Ekle
Silinecek değeri girin:
Eleman Sil
Hata
Aranan değer bulunamadı !
Tamam

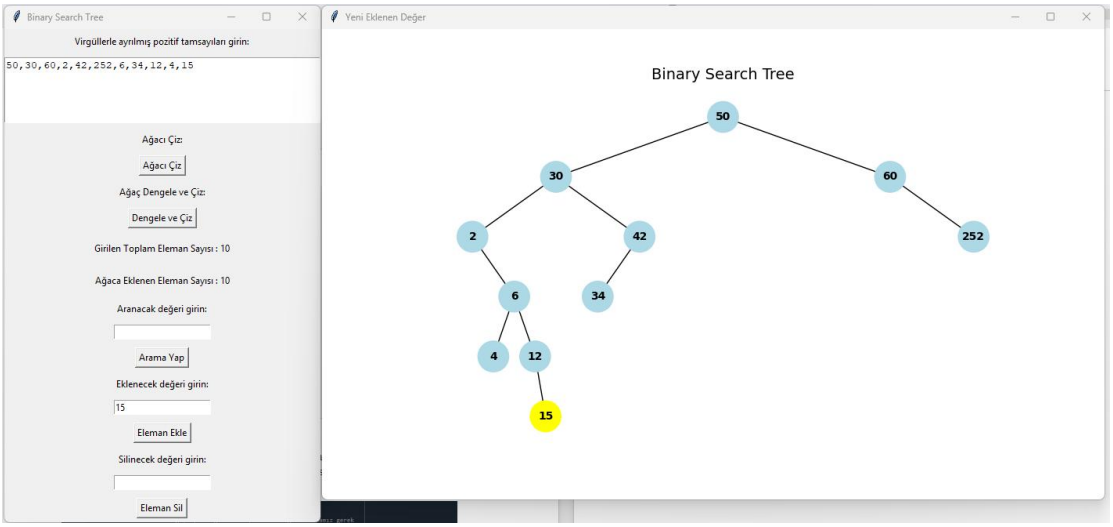
Ekleme İşlemi :



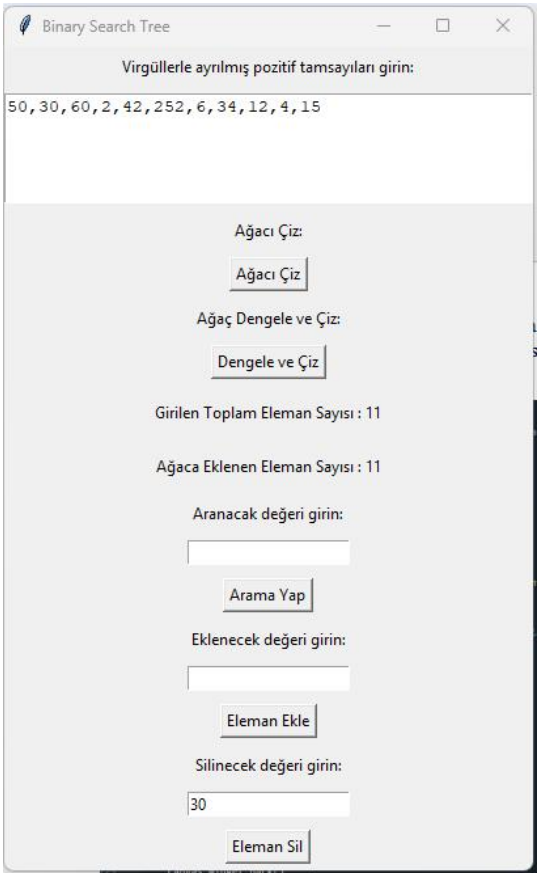
Kullanıcı eklenecek değeri girer ve butona basar. Buton Ekle() fonksiyonunu çağırır. Bu fonskiyon girilen değerin tipini ve ilk girilen ağaç dizisinin boş olup olmadığını kontrol eder.

```
86      def Ekle():
87          try:
88              #eklemeden önce ağaç için değer eklenip eklenmediğine bakmamız gerek
89              inputData = txt_inputDizi.get("1.0", tk.END).strip()
90              if(inputData == ""):
91                  messagebox.showerror("Hata","Ağaç Dizisi Boş")
92                  return
93
94              eklenecekDeger = int(txt_inputEklenecek.get())
95              inputData = txt_inputDizi.get("1.0", tk.END).strip()
96              lastData = inputData + "," + str(eklenecekDeger)
97              txt_inputDizi.delete("1.0", tk.END)
98              txt_inputDizi.insert("1.0", lastData)
99          except ValueError:
100             messagebox.showerror("Hata", "Lütfen geçerli bir sayı girin!")
101             return
102
103         if(len(numbers)>=100):
104             messagebox.showerror("Hata","Eleman Sayısı 100'den fazla olduğu için yeni eleman eklenemez !!")
105         else:
106             numbers.append(eklenecekDeger)
107
108         # başlangıçtaki ağaç dizisi
109         AgacDizisi = [0] *(2**25)
110
111         # aldığımız input listesini BST dizisine dönüştür
112         for number in numbers:
113             AgacDizisi = BSTDiziOlustur(AgacDizisi, number)
114
115         bulunanIndex = IndexAra(AgacDizisi, eklenecekDeger)
116
117         new_window = tk.Toplevel(root)
118         new_window.title("Yeni Eklenen Değer")
119
120         fig = DugumBoya(AgacDizisi, bulunanIndex,"yellow")
121         canvas = FigureCanvasTkAgg(fig, master=new_window)
122         canvas_widget = canvas.get_tk_widget()
123         canvas_widget.pack()
124         canvas.draw()
125         plt.close('all')
```

Bu fonksiyon girilen değeri numbers dizisinin sonuna ver kullanıcının ilk girdiği TextBox'a girilen değeri ekler. Daha Sonra BST oluşturur. Ağaç oluştuktan sonra yeni eklenen değer IndexAra() ile aranır. Yeni eklenen değerin indeksine ulaştıktan sonra DugumBoya() fonskiyonu ile eklenen elemanın indisini sarı renk ile boyarız. Diğer düğümler mavi olarak görüntülenir.



Silme Fonksiyonu :



Kullanıcı silinecek değeri TextBox'a girer ve butona basar. Buton Sil() fonksiyonunu çağırır.

```python
def Sil():
    try:
        silinecekDeger = int(txt_inputSilinecek.get())
    except ValueError:
        messagebox.showerror("Hata", "Lütfen geçerli bir sayı girin!")
        return

    # başlangıçtaki ağaç dizisi
    AgacDizisi = [0] *(2**25)
    # aldığımız input listesini BST dizisine dönüştür
    for number in numbers:
        AgacDizisi = BSTDiziOlustur(AgacDizisi, number)

    # input olarak alınan değerin indexini ara
    bulunanIndex = IndexAra(AgacDizisi, silinecekDeger)

    if(bulunanIndex==-1):
        messagebox.showerror("Hata","Aranan Değer Ağaçta Mevcut Değil")
        return
    else:
        #numbers listesinden değeri çıkar
        numbers.remove(silinecekDeger)
        #textboxı güncelleme
        updatedData = ",".join([str(num) for num in numbers])
        txt_inputDizi.delete("1.0", tk.END)
        txt_inputDizi.insert("1.0", updatedData)

    AgacDizisi=DugumSil(AgacDizisi,bulunanIndex)

    new_window = tk.Toplevel(root)
    new_window.title("Ağaç Güncellendi")

    fig = AgacCiz(AgacDizisi)  # Ağacı çiz
    canvas = FigureCanvasTkAgg(fig, master=new_window)
    canvas_widget = canvas.get_tk_widget()
    canvas_widget.pack()
    canvas.draw()
    plt.close('all')
```

Burada kullancının girdiği değer kontrol edilir. Hata durumunda ekrana bir messagebox çizilir. Daha sonra kullanıcının ilk girdiği dizi değeri bir AgacDizisi içerisine aktarılır. Sonrasında IndexAra() fonskiyonu kullanılarak değerin indeksi bulunur. Eğer değer bulunamazsa ekrana bir messagebox çizilir. Eğer değer bulunursa Bu değer he mnumbers dizisinden hem de kullanıcının ilk giriş yaptığı textboxtan kaldırılır. Daha sonra DugumSil() fonksiyonu çağırılır. Bu fonskiyon **AgacMetotları.py** içerisinde bulunur.

```python
def DugumSil(AgacDizisi, bulunanIndex):
    idx=bulunanIndex

    left_child = 2 * idx + 1
    right_child = 2 * idx + 2

   # Yaprak düğümse (çocukları yoksa), direkt sil
    if (left_child <= len(AgacDizisi) and AgacDizisi[left_child] == 0) and (right_child <= len(AgacDizisi) and AgacDizisi[right_child] == 0):
        AgacDizisi[idx] = 0
        return AgacDizisi

    # Tek çocuğu olan düğüm
    if (right_child <= len(AgacDizisi) and AgacDizisi[right_child] == 0):# Sadece sol çocuk var
        sol_sol_cocuk =2*left_child+1 #sol çocuk eğer bir ağaçsa bu ağacı yukarı taşımamız gerekiyor
        sol_sag_cocuk =2*left_child+2
        if(sol_sol_cocuk<len(AgacDizisi) and sol_sag_cocuk<len(AgacDizisi) and AgacDizisi[sol_sol_cocuk]!=0 and AgacDizisi[sol_sag_cocuk]!=0):
            AltAgacTasi(AgacDizisi, left_child, idx)
            return AgacDizisi
        else:
            AgacDizisi[idx] = AgacDizisi[left_child]
            DugumSil(AgacDizisi, left_child)  #sol çocuk bir ağaç değilse sadece sol çocuğu taşı

    elif (left_child <= len(AgacDizisi) and AgacDizisi[left_child] == 0):  # Sadece sağ çocuk var
        sag_sol_cocuk =2*right_child+1 #sag çocuk eğer bir ağaçsa bu ağacı yukarı taşımamız gerekiyor
        sag_sag_cocuk =2*right_child+2
        if(sag_sol_cocuk<len(AgacDizisi) and sag_sag_cocuk<len(AgacDizisi) and AgacDizisi[sag_sol_cocuk]!=0 and AgacDizisi[sag_sag_cocuk]!=0):
            AltAgacTasi(AgacDizisi, right_child, idx)
            return AgacDizisi
        else:
            AgacDizisi[idx] = AgacDizisi[right_child]
            DugumSil(AgacDizisi, right_child)  # sağ çocuğu sil
    else:
        # İki çocuğu varsa: sol alt ağaçtan en büyük değeri al
        min_in_left = left_child
        while 2 * min_in_left + 2 < len(AgacDizisi) and AgacDizisi[2 * min_in_left + 2] != 0:
            min_in_left = 2 * min_in_left + 2

        # Bulunan değeri düğümün yerine koy ve sil
        AgacDizisi[idx] = AgacDizisi[min_in_left]
        DugumSil(AgacDizisi, min_in_left)  # Min düğümü sil

    return AgacDizisi
```

Burada öncelikle yollanan düğümün sol ve sağ çocuklarının indisi hesaplanır.

Daha sonra bu düğüm yaprak düğüm mü kontrolü yapılır. Bu kontrol sağ ve sol çocuklarının indisinin AgacDizisi'nin boyutuna ve Cocuklarının değerinin boş mu dolu mu olduğuna göre kotnrol edilir. Eğer yaprak düğümse düğüm silinir.

Eğer Yaprak düğüm değilse iki çocuğu mu yoksa bir çocuğu mu olup olmadığı durumunu kontrol etmemiz gerekiyor. Eğer sadece sol veya sağ çocuğu varsa ve bu çocuk bir ağaç değilse bu düğümü silinen düğüm yerine atıyoruz ve fonksiyonu tekrardan çağırıp diğer düğümler için de tekrar silme işlemini başaltıyoruz.

Eğer sadece sol veya sağ çocuğu varsa ve bu süğüm bir ağaçsa bu ağacı yukarı taşımamız gerekiyor. Bu taşıma işlemini AltAgacTasi() fonksiyonu ile yapıyoruz. Bu fonksiyon AgacMetotları.py içerisinde bulunuyor.

```
154     def AltAgacTasi(AgacDizisi, kaynakIndex, hedefIndex,sifirla=0):
155         #alt ağacı yukarı taşıma
156 
157         #bu kontrolü yapmamızdaki amaç fonksiyon kendini tekrarlı çağırdığı için
158         #en son adımda hedef indisi 0 olarak setleme sorununun önüne geçmek
159         if kaynakIndex >= len(AgacDizisi) or AgacDizisi[kaynakIndex] == 0:
160             return
161 
162         AgacDizisi[hedefIndex] = AgacDizisi[kaynakIndex]
163 
164         # taşınacak indisler ve taşınılacak indisler
165         kaynak_sol_cocuk = 2 * kaynakIndex + 1
166         kaynak_sag_cocuk = 2 * kaynakIndex + 2
167         hedef_sol_cocuk = 2 * hedefIndex + 1
168         hedef_sag_cocuk = 2 * hedefIndex + 2
169 
170         # Sol alt ağacı taşı
171         if kaynak_sol_cocuk < len(AgacDizisi) and AgacDizisi[kaynak_sol_cocuk] != 0:
172             AltAgacTasi(AgacDizisi, kaynak_sol_cocuk, hedef_sol_cocuk,1)
173 
174         # Sağ alt ağacı taşı
175         if kaynak_sag_cocuk < len(AgacDizisi) and AgacDizisi[kaynak_sag_cocuk] != 0:
176             AltAgacTasi(AgacDizisi, kaynak_sag_cocuk, hedef_sag_cocuk,1)
177 
178         if sifirla==1:
179             AgacDizisi[kaynakIndex] = 0
```

Bu fonksiyon yine BST indis hesaplarını yaprak kendi tekrarlı şekilde çağırarak yavru ağacı silinecek indise doğru taşır. Burada ilk durumda 0 olmasının nedeni taşınan son çağırım olduğunda taşınan ağacın sol çocuğunu 0 yapma sorununun önüne geçmemizi sağlıyor. Kendi içindeki tekrarlı çağırımalrda bu değeri 1 olarak gönderiyoruz ve taşıma işlemi başarılı bir şekilde sonuçlanıyor.

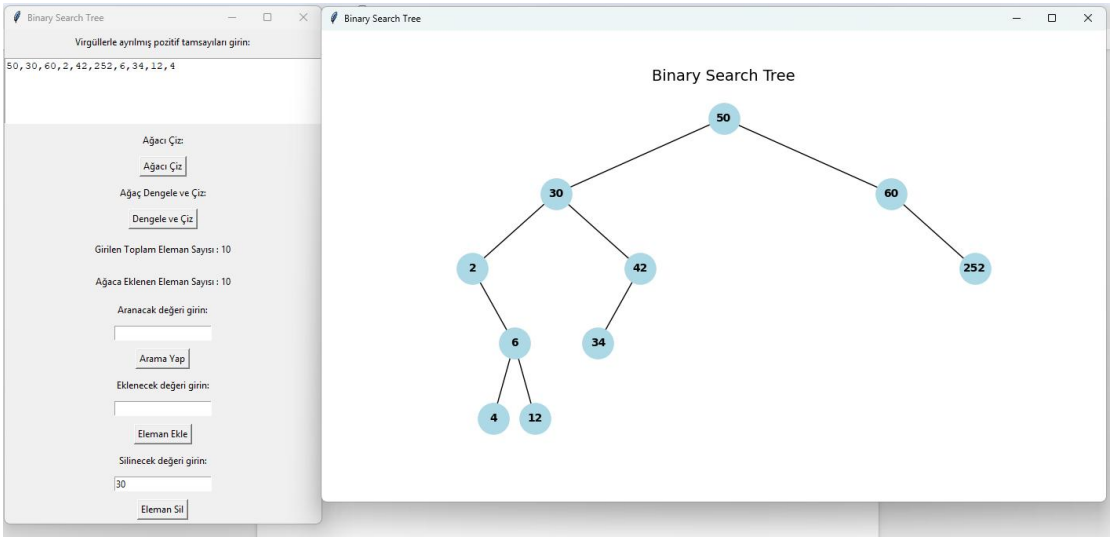
Son olarak eğer silinecek düğümün iki çocuğu varsa soldaki düğümün en büyük çocuğu silinecek indise taşıyoruz.

Silme işlemi bittikten sonra tekrardan oluşan ağacı yeni bir pencerede çizdiriyoruz.

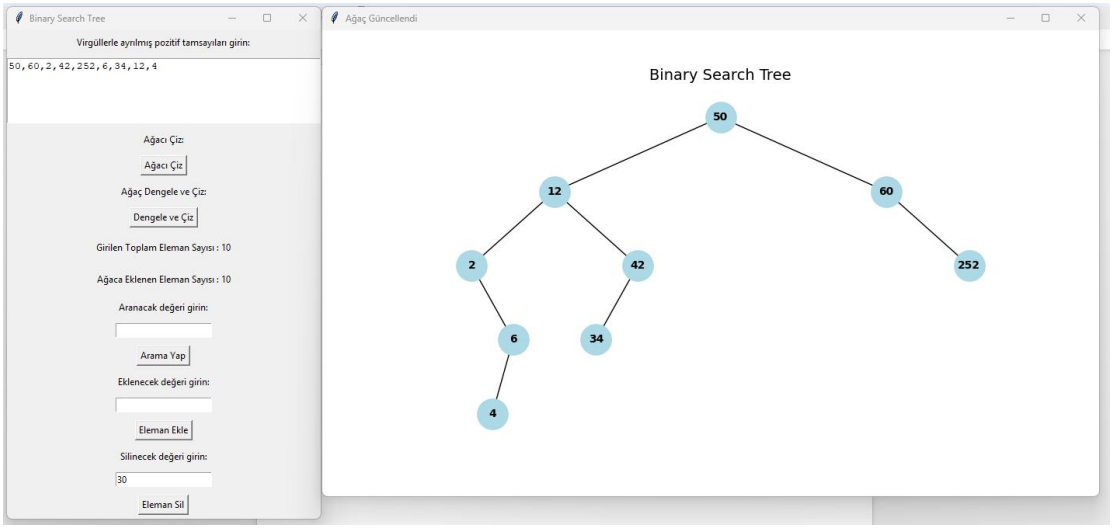
Çıktılar :

1. Çıktı

Silme işlemi öncesi :

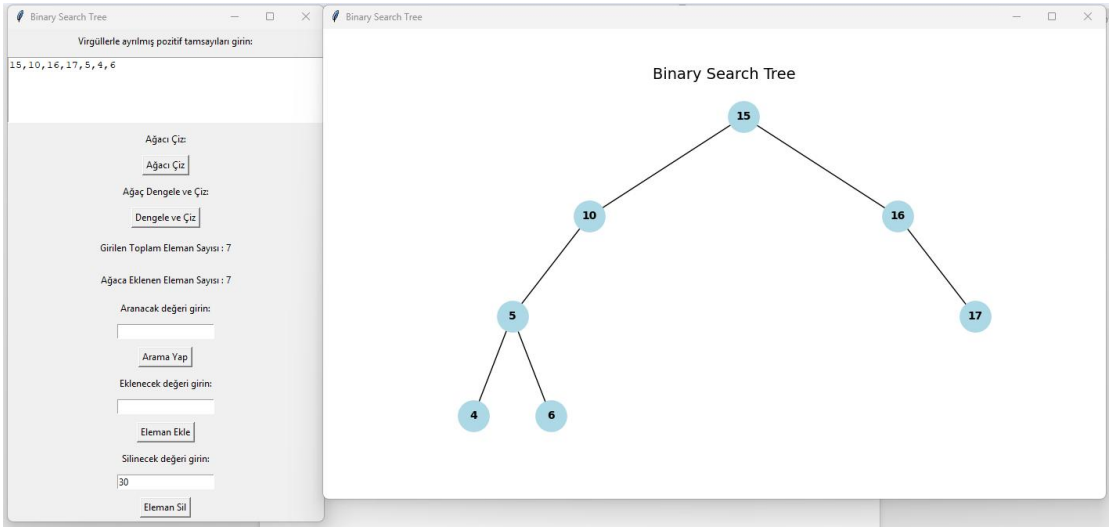


Silme işlemi sonrası :

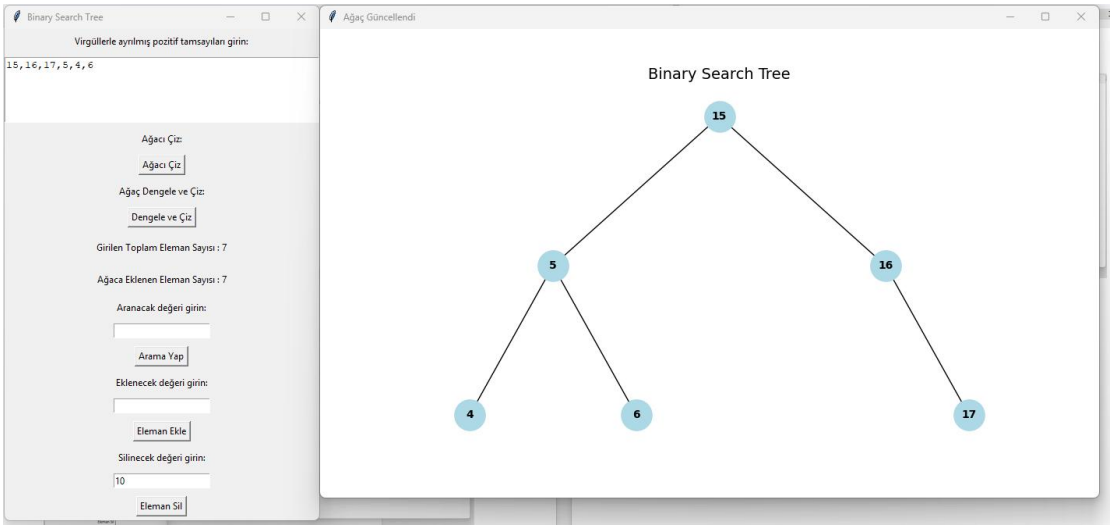


2. Çıktı :

Silme işlemi öncesi :



Silme işlemi sonrası :



Hata Durumu :