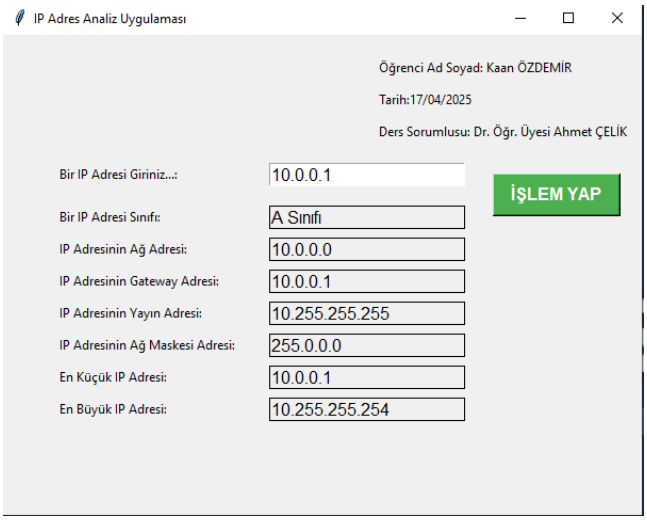
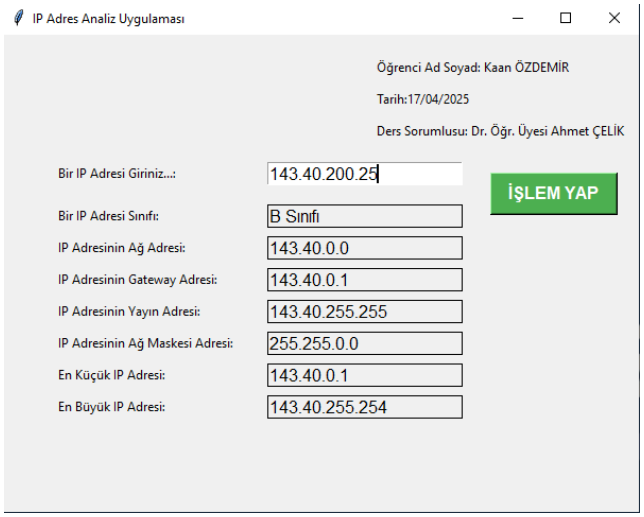
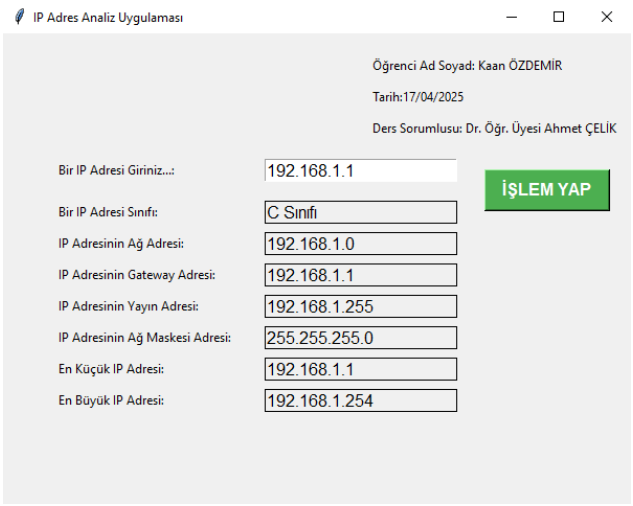
Öğrenci Adı Soyadı: Kaan ÖZDEMİR

Numarası: 202485151010

IPV4 IP ADRESLERİNİN SINIFLANDIRILMASI







Uygulama Kaynak Kodu:

```
import tkinter as tk  # Tkinter kütüphanesini içe aktarır.
from tkinter import messagebox  # Messagebox modülünü içe aktarır.
import ipaddress  # ipaddress modülünü içe aktarır.


class IPAnalyzerApp:
    def __init__(self, root):
        self.root = root  # Tkinter root penceresini saklar.
        self.root.title("IP Adres Analiz Uygulaması")  # Pencere başlığını ayarlar.
        self.root.geometry("600x450")  # Pencere boyutunu ayarlar.

        # Sonuç değişkenleri sözlüğünü başlatır.
        self.result_vars = {}

        # Başlık bilgileri
        tk.Label(root, text="Öğrenci Ad Soyad: Kaan ÖZDEMİR").place(x=350, y=20)  # Öğrenci ad soyad etiketini oluşturur.
        tk.Label(root, text="Tarih:17/04/2025").place(x=350, y=50)  # Tarih etiketini oluşturur.
        tk.Label(root, text="Ders Sorumlusu: Dr. Öğr. Üyesi Ahmet ÇELİK").place(x=350, y=80)  # Ders sorumlusu etiketini
oluşturur.

        # IP giriş alanı
        tk.Label(root, text="Bir IP Adresi Giriniz...:").place(x=50, y=120)  # IP adresi giriş etiketi oluşturur.
        self.ip_entry = tk.Entry(root, width=20, font=('Arial', 12))  # IP adresi giriş alanını oluşturur.
        self.ip_entry.place(x=250, y=120)  # IP adresi giriş alanını yerleştirir.

        # Sonuç alanları
        self.create_result_field("Bir IP Adresi Sınıfı:", 160)  # IP sınıfı sonuç alanını oluşturur.
        self.create_result_field("IP Adresinin Ağ Adresi:", 190)  # Ağ adresi sonuç alanını oluşturur.
        self.create_result_field("IP Adresinin Gateway Adresi:", 220)  # Gateway adresi sonuç alanını oluşturur.
        self.create_result_field("IP Adresinin Yayın Adresi:", 250)  # Yayın adresi sonuç alanını oluşturur.
        self.create_result_field("IP Adresinin Ağ Maskesi Adresi:", 280)  # Ağ maskesi adresi sonuç alanını oluşturur.
        self.create_result_field("En Küçük IP Adresi:", 310)  # En küçük IP adresi sonuç alanını oluşturur.
        self.create_result_field("En Büyük IP Adresi:", 340)  # En büyük IP adresi sonuç alanını oluşturur.

        # Buton
        self.analyze_btn = tk.Button(root, text="İŞLEM YAP", command=self.analyze_ip,
                                     bg='#4CAF50', fg='white', font=('Arial', 12, 'bold'))  # İşlem yap butonunu oluşturur.
        self.analyze_btn.place(x=460, y=130, width=120, height=40)  # İşlem yap butonunu yerleştirir.

    def create_result_field(self, label_text, y_pos):
        """Sonuç etiketi ve giriş alanını oluşturur."""
        tk.Label(self.root, text=label_text).place(x=50, y=y_pos)  # Sonuç etiketini oluşturur.
        result_var = tk.StringVar()  # Sonuç değişkenini oluşturur.
        result_var.set("")  # Sonuç değişkenini boş dizeyle başlatır.
        entry = tk.Entry(self.root, textvariable=result_var, state='readonly',
                         font=('Arial', 12), width=20, relief='solid')  # Sonuç giriş alanını oluşturur.
        entry.place(x=250, y=y_pos)  # Sonuç giriş alanını yerleştirir.
        self.result_vars[label_text] = result_var  # Sonuç değişkenini sözlüğe ekler.

    def analyze_ip(self):
        """Girilen IP adresini analiz eder ve sonuçları görüntüler."""
        ip_address = self.ip_entry.get().strip()  # Girilen IP adresini alır ve boşlukları temizler.

        if not ip_address:
            messagebox.showerror("Hata", "Lütfen bir IP adresi giriniz!")  # Hata mesajı görüntüler.
            return

        try:
            # IP adresini kontrol et
            ip = ipaddress.IPv4Address(ip_address)  # Girilen adresi IPv4 adresi olarak doğrular.
            network = ipaddress.IPv4Network(f"{ip}/{self.get_netmask(ip)}", strict=False)  # IP adresi ve netmask ile ağı
oluşturur.

            # Sonuçları hesapla
            self.result_vars["Bir IP Adresi Sınıfı:"].set(self.get_ip_class(ip))  # IP sınıfını hesaplar ve görüntüler.
```

```
            self.result_vars["IP Adresinin Ağ Adresi:"].set(str(network.network_address))  # Ağ adresini alır ve görüntüler.
            self.result_vars["IP Adresinin Gateway Adresi:"].set(self.get_gateway_address(network))  # Gateway adresini alır ve görüntüler.
            self.result_vars["IP Adresinin Yayın Adresi:"].set(str(network.broadcast_address))  # Yayın adresini alır ve görüntüler.
            self.result_vars["IP Adresinin Ağ Maskesi Adresi:"].set(str(network.netmask))  # Ağ maskesini alır ve görüntüler.
            self.result_vars["En Küçük IP Adresi:"].set(self.get_min_host(network))  # En küçük IP adresini alır ve görüntüler.
            self.result_vars["En Büyük IP Adresi:"].set(self.get_max_host(network))  # En büyük IP adresini alır ve görüntüler.

        except ValueError as e:
            messagebox.showerror("Hata", f"Geçersiz IP adresi: {str(e)}")  # Geçersiz IP adresi hatası görüntüler.

    def get_ip_class(self, ip):
        """Verilen IP adresinin sınıfını belirler."""
        first_octet = int(str(ip).split('.')[0])  # IP adresinin ilk oktetini alır.

        if 1 <= first_octet <= 126:
            return "A Sınıfı"  # A sınıfı IP adresini döndürür.
        elif 128 <= first_octet <= 191:
            return "B Sınıfı"  # B sınıfı IP adresini döndürür.
        elif 192 <= first_octet <= 223:
            return "C Sınıfı"  # C sınıfı IP adresini döndürür.
        else:
            return "Bilinmeyen Sınıf"  # Bilinmeyen sınıf IP adresini döndürür.

    def get_netmask(self, ip):
        """Verilen IP adresine karşılık gelen netmask'ı döndürür."""
        first_octet = int(str(ip).split('.')[0])  # IP adresinin ilk oktetini alır.

        if 1 <= first_octet <= 126:
            return 8  # A sınıfı için /8 netmask'ı döndürür.
        elif 128 <= first_octet <= 191:
            return 16  # B sınıfı için /16 netmask'ı döndürür.
        elif 192 <= first_octet <= 223:
            return 24  # C sınıfı için /24 netmask'ı döndürür.
        else:
            return 32  # Diğer durumlar için /32 netmask'ı döndürür.

    def get_gateway_address(self, network):
        """Verilen ağın gateway adresini hesaplar."""
        # Genellikle ağdaki ilk kullanılabilir adres gateway olarak kullanılır
        return str(network.network_address + 1)  # Ağ adresine 1 ekleyerek gateway adresini bulur.

    def get_min_host(self, network):
        """Verilen ağdaki en küçük host adresini döndürür."""
        # Ağdaki ilk kullanılabilir host adresi (genellikle network_address + 1)
        hosts = list(network.hosts())  # Ağdaki host adreslerinin listesini alır.
        return str(hosts[0]) if hosts else "Yok"  # Liste boş değilse ilk hostu, değilse "Yok" döndürür.

    def get_max_host(self, network):
        """Verilen ağdaki en büyük host adresini döndürür."""
        # Ağdaki son kullanılabilir host adresi (genellikle broadcast_address - 1)
        hosts = list(network.hosts())  # Ağdaki host adreslerinin listesini alır.
        return str(hosts[-1]) if hosts else "Yok"  # Liste boş değilse son hostu, değilse "Yok" döndürür.


if __name__ == "__main__":
    root = tk.Tk()  # Bir Tkinter root penceresi oluşturur.
    app = IPAnalyzerApp(root)  # IPAnalyzerApp sınıfının bir örneğini oluşturur.
    root.mainloop()  # Tkinter olay döngüsünü başlatır.
```

IPv4 ve IPv6: İnternet Protokolleri

İnternet Protokolü (IP), bilgisayar ağlarında, özellikle de küresel internet üzerinde cihazların birbirini tanımasını ve iletişim kurmasını sağlayan temel teknolojidir. Herhangi bir ağa bağlı cihazın, veri paketlerini doğru hedefe yönlendirebilmek için benzersiz bir IP adresine sahip olması gerekir. Zaman içinde, internetin büyümesi ve teknolojik ihtiyaçların değişmesiyle IP'nin farklı sürümleri geliştirilmiştir. Bu sürümlerden en yaygın olarak bilinen ve kullanılan ikisi İnternet Protokolü sürüm 4 (IPv4) ve onun halefi olan İnternet Protokolü sürüm 6 (IPv6)'dır.

IPv4, internetin ilk yaygın kullanılan protokolüdür ve 32 bit'lik adres yapısıyla yaklaşık 4,3 milyar IP adresi sağlar. Adresler dört ondalık sayıdan oluşur (örnek: 192.168.1.1). IPv4 yaygın olarak kullanılmakta ancak adres yetersizliği yaşanmaktadır. IPv6, 128 bit'lik yapısıyla 2128 kadar adres sağlayarak adres sıkıntısını çözmek amacıyla geliştirilmiştir. Onaltılık formatta ve sekiz blok halinde yazılır (örnek: 2001:0db8:85a3:0000:0000:8a2e:0370:7334).

IPv4 ile IPv6 arasındaki temel farklar:

- IPv4 32 bit, IPv6 128 bit uzunluğundadır.
- IPv4 NAT gerektirirken IPv6 doğrudan adresleme sunar.
- IPv6, IPsec gibi güvenlik protokollerini destekler.
- IPv6'da broadcast yerine multicast ve anycast kullanılır.
- IPv6, SLAAC ile otomatik adres ataması yapabilir.

Avantajlar:

- IPv4: Yaygın uyumluluk, kısa adres formatı.
- IPv6: Büyük adres kapasitesi, gelişmiş güvenlik, NAT gereksinimi yok.

Dezavantajlar:

- IPv4: Adres yetersizliği, NAT gerekliliği, daha az güvenlik.
- IPv6: Uzun adres yapısı, eski cihazlarla uyumsuzluk, geçiş süreci karmaşık.

IPv6'nın kademeli geçiş sürecinde çift yığın (dual-stack) teknolojisi, cihazların hem IPv4 hem de IPv6 adreslerini destekleyerek uyumluluğu artırmada önemli bir rol oynamaktadır. Ayrıca, tünelleme (tunneling) gibi mekanizmalar, IPv6 trafiğinin IPv4 ağları üzerinden iletilmesine olanak tanıyarak geçişi kolaylaştırmaktadır.

Sonuç olarak, IPv6 geleceğin internet ihtiyaçlarına daha uygun bir protokol olup, IPv4 ile birlikte kademeli olarak yaygınlaşmaktadır."