Instituto Politécnico Nacional

Escuela Superior de Cómputo

“ESCOM”

Ingeniería en Sistemas Computacionales

Unidad de aprendizaje:

Redes de Computadoras

Proyecto 3

Calculadoras VLSM / CIDR

Integrantes:

Blanco Reséndiz Cuauhtémoc – 2024640579 – cblancor200@alumno.ipn.mx

Fecha de entrega: 5 de Enero de 2026

ÍNDICE

INTRODUCCIÓN

En el ámbito de las redes de computadoras, el diseño y administración eficiente de direcciones IP constituye uno de los pilares fundamentales para la implementación de infraestructuras de red escalables y optimizadas. La creciente demanda de dispositivos conectados a internet y la limitación de direcciones IPv4 han hecho imperativa la necesidad de técnicas avanzadas de subdivisión de redes que maximicen el aprovechamiento del espacio de direccionamiento disponible. La asignación manual de subredes presenta varios desafíos significativos:

1. **Desperdicio de direcciones**: frecuentemente resulta en asignaciones ineficientes donde subredes pequeñas consumen bloques de direcciones desproporcionadamente grandes.
2. **Escalabilidad limitada**: En redes corporativas con múltiples departamentos de diferentes tamaños, la asignación estática de subredes dificulta la adaptación a cambios organizacionales.
3. **Dificultad en la planificación**: Administradores de red sin herramientas adecuadas enfrentan retos significativos al planear expansiones futuras y realizar modificaciones a la infraestructura existente.

El objetivo es desarrollar dos herramientas computacionales integradas que automaticen y optimicen el proceso de diseño y asignación de subredes IP:

1. **Calculadora CIDR**: Implementar una herramienta que, dada una dirección IP y prefijo CIDR o máscara de red, calcule automáticamente todas las propiedades de la red, incluyendo dirección de red, dirección de broadcast, rango de hosts válidos, máscara en diferentes notaciones y representación binaria.
2. **Calculadora VLSM**: Diseñar e implementar un sistema que, tomando como entrada una red base y una lista de requerimientos por subred, aplique el algoritmo Variable Length Subnet Masking (VLSM) para asignar bloques contiguos de direcciones optimizando el uso del espacio disponible, evitando solapamientos y minimizando el desperdicio.

EXPLICACION Y DESARROLLO DE CALCULADORAS

Calculadora VLSM

La Calculadora VLSM representa un desafío algorítmico significativamente más complejo que la calculadora CIDR. Su desarrollo requirió no solo entender los principios de Variable Length Subnet Masking, sino también diseñar un algoritmo eficiente que optimice el uso del espacio de direccionamiento. El núcleo del sistema implementa una estrategia greedy (voraz) que, aunque no garantiza la solución óptima absoluta en todos los casos, produce asignaciones altamente eficientes para la mayoría de escenarios prácticos.

El punto de partida del algoritmo es la premisa fundamental de VLSM: diferentes subredes pueden tener diferentes tamaños según sus necesidades específicas, y estas subredes deben asignarse de manera contigua dentro del espacio de direccionamiento de la red principal. Esta capacidad de variar el tamaño de la máscara por subred es lo que diferencia VLSM del subnetting tradicional y permite un aprovechamiento mucho más eficiente de las direcciones IP.

El algoritmo comienza con una fase de preparación donde cada requerimiento de hosts se ajusta para incluir las dos direcciones reservadas (red y broadcast). Este ajuste es crítico porque el cálculo de bloques debe considerar el espacio total necesario, no solo los hosts utilizables. Por ejemplo, una subred que requiere 50 hosts realmente necesita un bloque que pueda acomodar al menos 52 direcciones (50 hosts + red + broadcast).

Para cada subred en la lista ordenada, el sistema calcula el tamaño mínimo de bloque que puede satisfacer el requerimiento. Este cálculo utiliza la función techo de logaritmo base 2: ceil(log2(hosts_necesarios)). El resultado indica la cantidad de bits necesarios en la porción de host, lo que directamente determina el prefijo de la subred (32 - bits_host) y el tamaño total del bloque (2^bits_host).

La asignación de direcciones se realiza manteniendo un puntero que indica la próxima dirección disponible dentro de la red base. Este puntero se incrementa con el tamaño de cada bloque asignado, asegurando contigüidad y evitando solapamientos.

Calculadora CIDR

El algoritmo implementado sigue una secuencia lógica que replica el proceso mental que un administrador de redes experimentado realizaría, pero con la precisión y velocidad de la computación. Cuando se recibe una dirección IP con su prefijo correspondiente, el sistema primero valida la corrección sintáctica y semántica de la entrada. Esta validación incluye verificación de formato, rangos permitidos en cada octeto y coherencia entre los diferentes componentes de la dirección.

Una vez validada la entrada, el sistema convierte la dirección IP y la máscara de red a su representación numérica entera de 32 bits. Esta conversión es crucial porque permite realizar operaciones aritméticas y bitwise de manera eficiente. La dirección de red se calcula aplicando una operación AND bitwise entre la dirección IP y la máscara, lo que efectivamente "apaga" los bits correspondientes a la porción de host. Este cálculo es fundamental porque determina el punto de inicio de la subred.

El cálculo de la dirección de broadcast representa el complemento lógico de la operación anterior. Mientras que la dirección de red marca el inicio, el broadcast marca el fin del bloque de direcciones. Se obtiene aplicando una operación OR bitwise entre la dirección de red y el wildcard (complemento de la máscara). Este valor es particularmente importante porque identifica la dirección reservada para comunicación con todos los dispositivos de la subred.

CODIGO FUENTE

Main.py

```python
from cidr_calculator import CalculadoraCIDR
from vlsm_calculator import CalculadoraVLSM
import os

def limpiar_pantalla():
    os.system('cls' if os.name == 'nt' else 'clear')

def mostrar_menu():
    print("\n" + "="*50)
    print("CALCULADORAS DE RED - CIDR y VLSM")
    print("="*50)
    print("1. Calculadora CIDR")
    print("2. Calculadora VLSM")
    print("3. Ejecutar casos de prueba")
    print("4. Acerca del proyecto")
    print("5. Salir")
    print("="*50)

def ejecutar_casos_prueba():
    print("\n" + "="*50)
    print("CASOS DE PRUEBA")
    print("="*50)

    # Caso de prueba CIDR
    print("\n1. CASO CIDR: 192.168.10.0/24")
    print("-" * 40)
    try:
        cidr = CalculadoraCIDR("192.168.10.0/24")
        resultados = cidr.calcular()
        cidr.imprimir_resultados(resultados)
    except Exception as e:
        print(f"Error: {e}")

    # Caso de prueba VLSM
    print("\n2. CASO VLSM: 192.168.0.0/24 con A=100, B=50, C=25, D=10")
    print("-" * 40)
    try:
        vlsm = CalculadoraVLSM()
        vlsm.configurar_red_base("192.168.0.0/24")
        vlsm.agregar_subred("A", 100)
        vlsm.agregar_subred("B", 50)
```

Ilustración 1. Main.py p1

```
        vlsm.agregar_subred("B", 50)
        vlsm.agregar_subred("C", 25)
        vlsm.agregar_subred("D", 10)
        vlsm.calcular_vlsm()
        vlsm.imprimir_resultados()
    except Exception as e:
        print(f"Error: {e}")

    # Caso límite VLSM
    print("\n3. CASO LÍMITE VLSM: Red pequeña con muchos hosts")
    print("-" * 40)
    try:
        vlsm2 = CalculadoraVLSM()
        vlsm2.configurar_red_base("192.168.0.0/30")  # Solo 4 direcciones
        vlsm2.agregar_subred("S1", 10)  # Requiere 12 direcciones
        vlsm2.calcular_vlsm()
        vlsm2.imprimir_resultados()
    except ValueError as e:
        print(f"Error esperado: {e}")

    input("\nPresione Enter para continuar...")

def mostrar_acerca():
    """Muestra información sobre el proyecto."""
    limpiar_pantalla()
    print("\n" + "="*50)
    print("ACERCA DEL PROYECTO")
    print("="*50)
    print("\nCalculadoras de Red - CIDR y VLSM")
    print("Proyecto 3: Redes de Computadoras")
    print("\nIntegrantes del equipo:")
    print("- Blanco Resendiz Cuauhtemoc - 2024630579")
    print("\nFuncionalidades:")
    print("• Calculadora CIDR completa")
    print("• Calculadora VLSM")
    print("• Validación de entradas")
    print("• Manejo de errores")
    print("• Interfaz de consola")
    print("\nInstrucciones:")
    print("1. Ejecutar: python main.py")
```

Ilustración 2. Main.py p2

```
82      print("3. Seguir las instrucciones en pantalla")
83      print("\n" + "="*50)
84      input("\nPresione Enter para continuar...")
85
86  def main():
87      """Función principal del programa."""
88      while True:
89          limpiar_pantalla()
90          mostrar_menu()
91
92          opcion = input("\nSeleccione una opción (1-5): ").strip()
93
94          if opcion == '1':
95              limpiar_pantalla()
96              cidr = CalculadoraCIDR()
97              cidr.ejecutar_desde_consola()
98
99          elif opcion == '2':
100             limpiar_pantalla()
101             vlsm = CalculadoraVLSM()
102             vlsm.ejecutar_desde_consola()
103
104         elif opcion == '3':
105             limpiar_pantalla()
106             ejecutar_casos_prueba()
107
108         elif opcion == '4':
109             mostrar_acerca()
110
111         elif opcion == '5':
112             limpiar_pantalla()
113             print("\n¡Gracias por usar las Calculadoras de Red!")
114             print("Hasta luego.\n")
115             break
116
117         else:
118             print("\nOpción inválida. Intente nuevamente.")
119             input("Presione Enter para continuar...")
```

Ilustración 3. Main.py p3

```
121 if __name__ == "__main__":
122     try:
123         main()
124     except KeyboardInterrupt:
125         print("\n\nPrograma interrumpido por el usuario.")
126     except Exception as e:
127         print(f"\nError inesperado: {e}")
```

Ilustración 4. Main.py p4

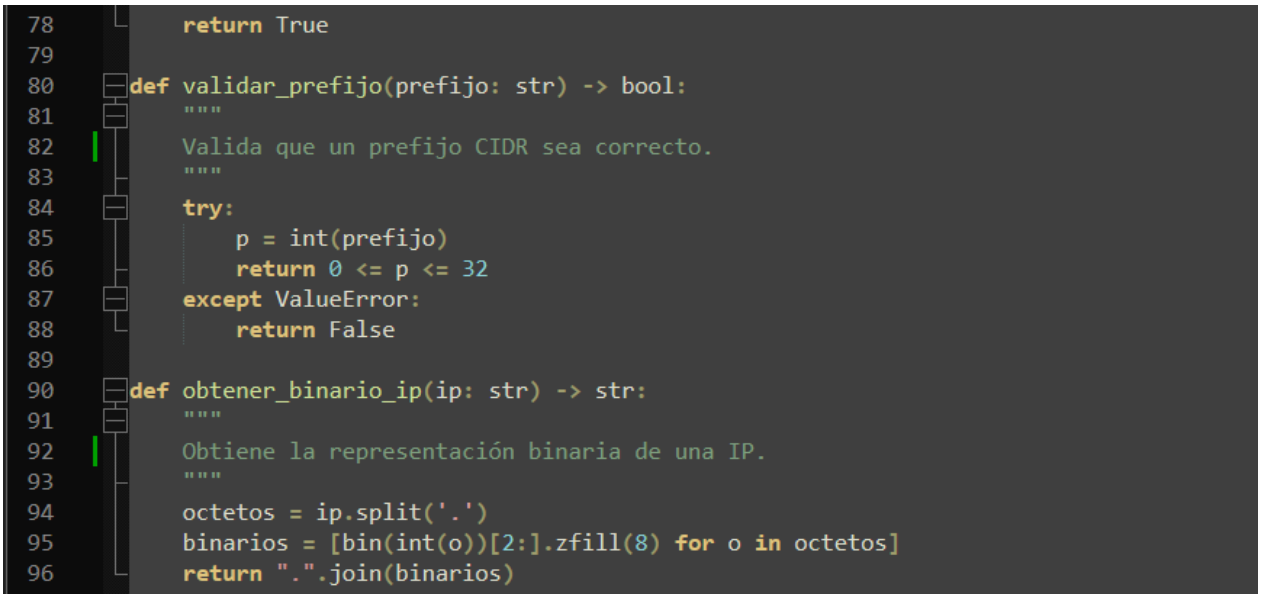
Ip_utils.py

```python
import re
import math

def ip_a_entero(ip: str) -> int:
    """
    Convierte una dirección IP en formato string a entero de 32 bits.
    """
    octetos = ip.split('.')
    if len(octetos) != 4:
        raise ValueError("La IP debe tener 4 octetos")

    resultado = 0
    for i, octeto in enumerate(octetos):
        valor = int(octeto)
        if valor < 0 or valor > 255:
            raise ValueError(f"Octeto {i+1} fuera de rango (0-255)")
        resultado = (resultado << 8) | valor

    return resultado

def entero_a_ip(entero: int) -> str:
    """
    Convierte un entero de 32 bits a dirección IP en formato string.
    """
    if entero < 0 or entero > 0xFFFFFFFF:
        raise ValueError("Entero fuera de rango para dirección IP")

    octetos = []
    for i in range(3, -1, -1):
        octeto = (entero >> (8 * i)) & 0xFF
        octetos.append(str(octeto))

    return ".".join(octetos)

def mascara_a_prefijo(mascara: str) -> int:
    """
    Convierte máscara en formato decimal punteado a prefijo CIDR.
    """
    entero = ip_a_entero(mascara)

```

Ilustración 5. ip_utils.py p1

```
    entero = ip_a_entero(mascara)

    # Verificar que sea una máscara válida
    binario = bin(entero)[2:].zfill(32)

    # Buscar transición de 1 a 0
    encontrado_cero = False
    for bit in binario:
        if bit == '0':
            encontrado_cero = True
        elif encontrado_cero and bit == '1':
            raise ValueError("Máscara inválida: los 1s deben ser contiguos")

    return binario.count('1')

def prefijo_a_mascara(prefijo: int) -> str:
    """
    Convierte prefijo CIDR a máscara en formato decimal punteado.
    """
    if prefijo < 0 or prefijo > 32:
        raise ValueError("Prefijo debe estar entre 0 y 32")

    mascara_entero = (0xFFFFFFFF << (32 - prefijo)) & 0xFFFFFFFF
    return entero_a_ip(mascara_entero)

def validar_ip(ip: str) -> bool:
    """
    Valida que una dirección IP sea correcta.
    """
    patron = r'^(\d{1,3})\.(\d{1,3})\.(\d{1,3})\.(\d{1,3})$'
    match = re.match(patron, ip)

    if not match:
        return False

    for grupo in match.groups():
        if int(grupo) > 255:
            return False
```

Ilustración 6. ip_utils.py p2



```
78      return True
79
80  def validar_prefijo(prefijo: str) -> bool:
81      """
82      Valida que un prefijo CIDR sea correcto.
83      """
84      try:
85          p = int(prefijo)
86          return 0 <= p <= 32
87      except ValueError:
88          return False
89
90  def obtener_binario_ip(ip: str) -> str:
91      """
92      Obtiene la representación binaria de una IP.
93      """
94      octetos = ip.split('.')
95      binarios = [bin(int(o))[2:].zfill(8) for o in octetos]
96      return ".".join(binarios)
```

Ilustración 7. ip_utils.py p3

Cidr_calculator.py

```python
from ip_utils import *
import math


class CalculadoraCIDR:
    """
    Calculadora para operaciones CIDR
    """

    def __init__(self, ip_con_prefijo: str = None):
        self.ip = None
        self.prefijo = None
        self.mascara = None

        if ip_con_prefijo:
            self.analizar_entrada(ip_con_prefijo)

    def analizar_entrada(self, entrada: str):
        entrada = entrada.strip()

        # Formato: IP/prefijo
        if '/' in entrada:
            partes = entrada.split('/')
            if len(partes) != 2:
                raise ValueError("Formato incorrecto. Use: IP/prefijo o IP máscara")

            ip = partes[0].strip()
            prefijo_str = partes[1].strip()

            if not validar_ip(ip):
                raise ValueError("Dirección IP inválida")

            try:
                prefijo = int(prefijo_str)
            except ValueError:
                raise ValueError("Prefijo debe ser un número")

            if not 0 <= prefijo <= 32:
                raise ValueError("Prefijo debe estar entre 0 y 32")
```

Ilustración 8. cidr_calculator.py p1

```
            self.ip = ip
            self.prefijo = prefijo
            self.mascara = prefijo_a_mascara(prefijo)

        # Formato: IP máscara
        else:
            partes = entrada.split()
            if len(partes) != 2:
                raise ValueError("Formato incorrecto. Use: IP/prefijo o IP máscara")

            ip = partes[0].strip()
            mascara = partes[1].strip()

            if not validar_ip(ip):
                raise ValueError("Dirección IP inválida")

            if not validar_ip(mascara):
                raise ValueError("Máscara inválida")

            try:
                prefijo = mascara_a_prefijo(mascara)
            except ValueError as e:
                raise ValueError(f"Máscara inválida: {e}")

            self.ip = ip
            self.prefijo = prefijo
            self.mascara = mascara

    def calcular(self):
        """
        Calcula todas las propiedades de la red CIDR.
        """
        if not self.ip or self.prefijo is None:
            raise ValueError("Debe proporcionar una IP y prefijo primero")

        ip_entero = ip_a_entero(self.ip)
        mascara_entero = ip_a_entero(self.mascara) if self.mascara else (0xFFFFFFFF << (32 - self.prefijo)) & 0xFFFFFFFF
```

Ilustración 9. cidr_calculator.py p2

```python
        # Calcular dirección de red
        red_entero = ip_entero & mascara_entero
        red = entero_a_ip(red_entero)

        # Calcular broadcast
        wildcard = mascara_entero ^ 0xFFFFFFFF
        broadcast_entero = red_entero | wildcard
        broadcast = entero_a_ip(broadcast_entero)

        # Calcular primer y último host
        if self.prefijo == 32:
            primer_host = red
            ultimo_host = red
            hosts_validos = 1
        elif self.prefijo == 31:
            # Caso especial: /31 (RFC 3021)
            primer_host = red
            ultimo_host = broadcast
            hosts_validos = 2
        else:
            primer_host_entero = red_entero + 1
            ultimo_host_entero = broadcast_entero - 1
            primer_host = entero_a_ip(primer_host_entero)
            ultimo_host = entero_a_ip(ultimo_host_entero)
            hosts_validos = ultimo_host_entero - primer_host_entero + 1

        # Calcular número total de hosts
        if self.prefijo == 32:
            total_hosts = 1
        else:
            total_hosts = 2 ** (32 - self.prefijo)

        # Calcular desperdicio
        desperdicio = total_hosts - hosts_validos if self.prefijo < 31 else 0

        # Representaciones binarias
        bin_ip = obtener_binario_ip(self.ip)
        bin_mascara = obtener_binario_ip(self.mascara)
        bin_red = obtener_binario_ip(red)
```

Ilustración 10. cidr_calculator.py p3

```
        return {
            "ip_original": self.ip,
            "prefijo": self.prefijo,
            "mascara": self.mascara,
            "red": red,
            "broadcast": broadcast,
            "primer_host": primer_host,
            "ultimo_host": ultimo_host,
            "hosts_validos": hosts_validos,
            "total_hosts": total_hosts,
            "desperdicio": desperdicio,
            "bin_ip": bin_ip,
            "bin_mascara": bin_mascara,
            "bin_red": bin_red,
            "rango_hosts": f"{primer_host} - {ultimo_host}",
            "notacion_cidr": f"{red}/{self.prefijo}"
        }

    def imprimir_resultados(self, resultados: dict):
        print("\n" + "="*50)
        print("RESULTADOS CALCULADORA CIDR")
        print("="*50)
        print(f"IP original:      {resultados['ip_original']}")
        print(f"Notación CIDR:    {resultados['notacion_cidr']}")
        print(f"Máscara:          {resultados['mascara']} (/{resultados['prefijo']})")
        print(f"Dirección de red: {resultados['red']}")
        print(f"Broadcast:        {resultados['broadcast']}")
        print(f"Rango de hosts:   {resultados['rango_hosts']}")
        print(f"Hosts válidos:    {resultados['hosts_validos']}")
        print(f"Total direcciones: {resultados['total_hosts']}")
        print(f"Desperdicio:      {resultados['desperdicio']} direcciones")
        print("-"*50)
        print("REPRESENTACIÓN BINARIA")
        print(f"IP:       {resultados['bin_ip']}")
        print(f"Máscara:  {resultados['bin_mascara']}")
        print(f"Red:      {resultados['bin_red']}")
        print("="*50 + "\n")
```

Ilustración 11. cidr_calculator.py p4

```python
    def ejecutar_desde_consola(self):
        """
        Interfaz de consola para la calculadora CIDR.
        """
        print("\n" + "="*50)
        print("CALCULADORA CIDR")
        print("="*50)
        print("Formato de entrada:")
        print("1. IP/prefijo (ej: 192.168.1.0/24)")
        print("2. IP máscara (ej: 192.168.1.0 255.255.255.0)")
        print("="*50)

        while True:
            entrada = input("\nIngrese dirección IP con prefijo o máscara (o 'salir'): ").strip()

            if entrada.lower() == 'salir':
                break

            try:
                self.analizar_entrada(entrada)
                resultados = self.calcular()
                self.imprimir_resultados(resultados)
            except ValueError as e:
                print(f"Error: {e}")
                print("Intente nuevamente.")

if __name__ == "__main__":
    calculadora = CalculadoraCIDR()
    calculadora.ejecutar_desde_consola()
```

Ilustración 12. cidr_calculator.py p5

Vlsm_calculator.py

```python
from ip_utils import *
import math


class CalculadoraVLSM:
    """
    Calculadora para VLSM (Variable Length Subnet Masking).
    """

    def __init__(self):
        self.red_base = None
        self.prefijo_base = None
        self.subredes = []
        self.resultados = []

    def configurar_red_base(self, red_base: str):
        """
        Configura la red base para el cálculo VLSM.
        """
        if '/' not in red_base:
            raise ValueError("Formato incorrecto. Use: IP/prefijo (ej: 10.0.0.0/16)")

        partes = red_base.split('/')
        ip = partes[0].strip()
        prefijo_str = partes[1].strip()

        if not validar_ip(ip):
            raise ValueError("Dirección IP inválida")

        try:
            prefijo = int(prefijo_str)
        except ValueError:
            raise ValueError("Prefijo debe ser un número")

        if not 0 <= prefijo <= 32:
            raise ValueError("Prefijo debe estar entre 0 y 32")

        self.red_base = ip
        self.prefijo_base = prefijo

```

Ilustración 13. vlsm_calculator.py p1

```python
    def agregar_subred(self, nombre: str, hosts_requeridos: int):
        """
        Agrega una subred a la lista de requerimientos.
        """
        if hosts_requeridos <= 0:
            raise ValueError("El número de hosts debe ser mayor a 0")

        self.subredes.append({
            "nombre": nombre,
            "hosts_requeridos": hosts_requeridos
        })

    def calcular_vlsm(self):
        """
        Ejecuta el algoritmo VLSM y calcula las asignaciones.
        """
        if not self.red_base or self.prefijo_base is None:
            raise ValueError("Debe configurar la red base primero")

        if not self.subredes:
            raise ValueError("Debe agregar al menos una subred")

        # 1. Preparar lista: calcular hosts_necesarios (hosts + 2)
        for subred in self.subredes:
            subred["hosts_necesarios"] = subred["hosts_requeridos"] + 2

        # 2. Ordenar subredes por tamaño descendente
        subredes_ordenadas = sorted(self.subredes,
                                    key=lambda x: x["hosts_necesarios"],
                                    reverse=True)

        # 3. Inicializar puntero
        ip_actual_entero = ip_a_entero(self.red_base)
        espacio_total = 2 ** (32 - self.prefijo_base)
        limite_superior = ip_actual_entero + espacio_total

        self.resultados = []
```

Ilustración 14. vlsm_calculator.py p2

```python
        # 4. Asignar subredes
        for subred in subredes_ordenadas:
            # Calcular bits de host necesarios
            bits_host = math.ceil(math.log2(subred["hosts_necesarios"]))
            bits_red = 32 - bits_host

            # Calcular tamaño del bloque
            bloque = 2 ** bits_host

            # Verificar que haya espacio suficiente
            if ip_actual_entero + bloque > limite_superior:
                raise ValueError(
                    f"No hay espacio suficiente para la subred '{subred['nombre']}'. "
                    f"Espacio insuficiente en la red base."
                )

            # Calcular propiedades de la subred
            red = entero_a_ip(ip_actual_entero)
            broadcast_entero = ip_actual_entero + bloque - 1
            broadcast = entero_a_ip(broadcast_entero)

            # Calcular primer y último host (excepto para /31 y /32)
            if bits_red == 32:  # /32
                primer_host = red
                ultimo_host = red
                hosts_validos = 1
            elif bits_red == 31:  # /31 (RFC 3021)
                primer_host = red
                ultimo_host = broadcast
                hosts_validos = 2
            else:
                primer_host_entero = ip_actual_entero + 1
                ultimo_host_entero = broadcast_entero - 1
                primer_host = entero_a_ip(primer_host_entero)
                ultimo_host = entero_a_ip(ultimo_host_entero)
                hosts_validos = ultimo_host_entero - primer_host_entero + 1
```

Ilustración 15. vlsm_calculator.py p3

```
            # Calcular desperdicio
            desperdicio = bloque - hosts_validos if bits_red < 31 else 0

            # Guardar resultados
            self.resultados.append({
                "nombre": subred["nombre"],
                "hosts_requeridos": subred["hosts_requeridos"],
                "red": red,
                "prefijo": bits_red,
                "mascara": prefijo_a_mascara(bits_red),
                "broadcast": broadcast,
                "primer_host": primer_host,
                "ultimo_host": ultimo_host,
                "hosts_validos": hosts_validos,
                "tam_bloque": bloque,
                "desperdicio": desperdicio,
                "rango_hosts": f"{primer_host} - {ultimo_host}",
                "notacion_cidr": f"{red}/{bits_red}"
            })

            # Actualizar puntero
            ip_actual_entero += bloque

        # 5. Calcular espacio restante
        self.espacio_usado = ip_actual_entero - ip_a_entero(self.red_base)
        self.espacio_total = espacio_total
        self.espacio_restante = espacio_total - self.espacio_usado

        return self.resultados

    def imprimir_resultados(self):
        """
        Imprime los resultados del cálculo VLSM
        """
        if not self.resultados:
            print("No hay resultados para mostrar. Ejecute calcular_vlsm() primero.")
            return
```

Ilustración 16. vlsm_calculator.py p4

```
        print("\n" + "="*70)
        print("RESULTADOS CALCULADORA VLSM")
        print("="*70)
        print(f"Red base: {self.red_base}/{self.prefijo_base}")
        print(f"Espacio total: {self.espacio_total} direcciones")
        print(f"Espacio usado: {self.espacio_usado} direcciones")
        print(f"Espacio restante: {self.espacio_restante} direcciones")
        print("="*70)

        for i, subred in enumerate(self.resultados, 1):
            print(f"\nSubred {i}: {subred['nombre']}")
            print("-" * 40)
            print(f"  Hosts requeridos:  {subred['hosts_requeridos']}")
            print(f"  Dirección de red:  {subred['notacion_cidr']}")
            print(f"  Máscara:           {subred['mascara']} (/{subred['prefijo']})")
            print(f"  Rango de hosts:    {subred['rango_hosts']}")
            print(f"  Broadcast:         {subred['broadcast']}")
            print(f"  Hosts válidos:     {subred['hosts_validos']}")
            print(f"  Tamaño de bloque:  {subred['tam_bloque']} direcciones")
            print(f"  Desperdicio:       {subred['desperdicio']} direcciones")

        print("\n" + "="*70)
        print("RESUMEN")
        print("="*70)

        # Tabla resumen
        print(f"\n{'No.':<4} {'Subred':<15} {'Red':<18} {'Hosts Req.':<12} {'Hosts Disp.':<12} {'Desperdicio':<12}")
        print("-" * 75)

        for i, subred in enumerate(self.resultados, 1):
            print(f"{i:<4} {subred['nombre']:<15} {subred['notacion_cidr']:<18} "
                  f"{subred['hosts_requeridos']:<12} {subred['hosts_validos']:<12} "
                  f"{subred['desperdicio']:<12}")

        print("-" * 75)
        print(f"\nTotal hosts requeridos: {sum(s['hosts_requeridos'] for s in self.resultados)}")
        print(f"Total hosts disponibles: {sum(s['hosts_validos'] for s in self.resultados)}")
        print(f"Total desperdicio: {sum(s['desperdicio'] for s in self.resultados)} direcciones")
        print("="*70 + "\n")
```

Ilustración 17. vlsm_calculator.py p5

```python
    def ejecutar_desde_consola(self):
        """
        Interfaz de consola para la calculadora VLSM.
        """
        print("\n" + "="*50)
        print("CALCULADORA VLSM")
        print("="*50)

        # Configurar red base
        while True:
            red_base = input("Ingrese red base (ej: 192.168.0.0/24): ").strip()
            try:
                self.configurar_red_base(red_base)
                break
            except ValueError as e:
                print(f"Error: {e}")

        # Configurar subredes
        self.subredes = []
        print("\nIngrese las subredes (nombre y número de hosts requeridos)")
        print("Ingrese 'fin' cuando termine")

        while True:
            nombre = input("\nNombre de la subred (o 'fin'): ").strip()
            if nombre.lower() == 'fin':
                if not self.subredes:
                    print("Debe agregar al menos una subred.")
                    continue
                break

            try:
                hosts = input(f"Número de hosts para '{nombre}': ").strip()
                hosts_int = int(hosts)
                self.agregar_subred(nombre, hosts_int)
                print(f"Subred '{nombre}' agregada con {hosts_int} hosts.")
            except ValueError as e:
                print(f"Error: {e}")
```

Ilustración 18. vlsm_calculator.py p6

```python
            # Calcular VLSM
            try:
                print("\nCalculando asignaciones VLSM...")
                self.calcular_vlsm()
                self.imprimir_resultados()
            except ValueError as e:
                print(f"\nError en el cálculo: {e}")

if __name__ == "__main__":
    calculadora = CalculadoraVLSM()
    calculadora.ejecutar_desde_consola()
```

Ilustración 19. vlsm_calculator.py p7

INSTRUCCIONES DE INSTALACION

Método 1

Al descargar todos los archivos se encontrará la carpeta dist, dentro de esta encontraremos un ejecutable “main” , al dar doble click sobre este ejecutara el programa.

Método 2 (recomendado)

Descargar todos los archivos. Desde la terminal ejecutar el programa directamente con el comando Python main.py, de esta forma accederemos al menú principal de las calculadoras.

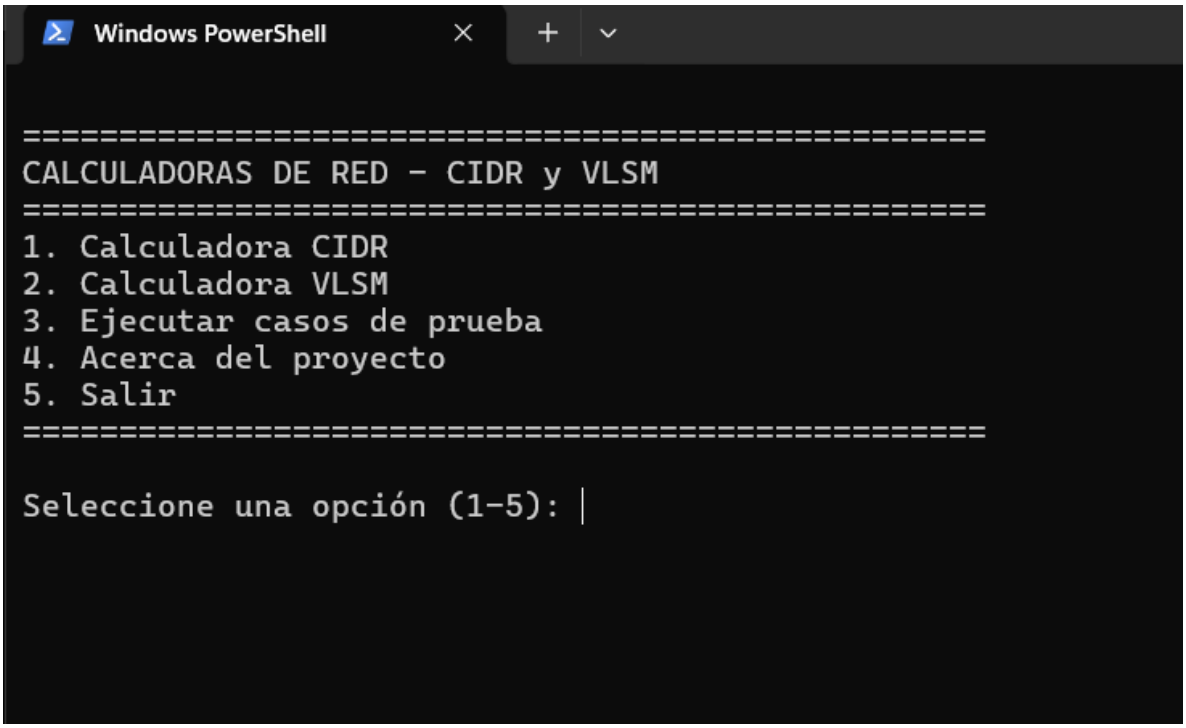


Ilustración 20. Menú principal.

PRUEBAS

Dentro del menú principal y para realizar las pruebas correspondientes de forma más fácil. Se encuentra el apartado 3. Ejecutar casos de prueba. Al ejecutarlo podremos ver los resultados de los casos de prueba solicitados por el profesor. Los cuales son los siguientes:

Caso 1 (CIDR):

Entrada: 10.0.0.0/8

Red: 10.0.0.0

Broadcast: 10.255.255.255

Hosts: 16,777,214

```
1. CASO CIDR: 192.168.10.0/24
----------------------------------------

================================================
RESULTADOS CALCULADORA CIDR
================================================
IP original:      192.168.10.0
Notación CIDR:    192.168.10.0/24
Máscara:          255.255.255.0 (/24)
Dirección de red: 192.168.10.0
Broadcast:        192.168.10.255
Rango de hosts:   192.168.10.1 - 192.168.10.254
Hosts válidos:    254
Total direcciones: 256
Desperdicio:      2 direcciones
------------------------------------------------
REPRESENTACIÓN BINARIA
IP:       11000000.10101000.00001010.00000000
Máscara:  11111111.11111111.11111111.00000000
Red:      11000000.10101000.00001010.00000000
================================================
```

Ilustración 21. Resultados caso 1.

Caso 2 (VLSM):

Red base: 172.16.0.0/16

Subredes: Ventas=500, TI=250, RH=120, Marketing=60

```
2. CASO VLSM: 192.168.0.0/24 con A=100, B=50, C=25, D=10
--------------------------------------

======================================================================
RESULTADOS CALCULADORA VLSM
======================================================================
Red base: 192.168.0.0/24
Espacio total: 256 direcciones
Espacio usado: 240 direcciones
Espacio restante: 16 direcciones
======================================================================

Subred 1: A
--------------------------------------
  Hosts requeridos:  100
  Dirección de red:  192.168.0.0/25
  Máscara:           255.255.255.128 (/25)
  Rango de hosts:    192.168.0.1 - 192.168.0.126
  Broadcast:         192.168.0.127
  Hosts válidos:     126
  Tamaño de bloque:  128 direcciones
  Desperdicio:       2 direcciones
```

Ilustración 22. Resultados caso 2 p1.

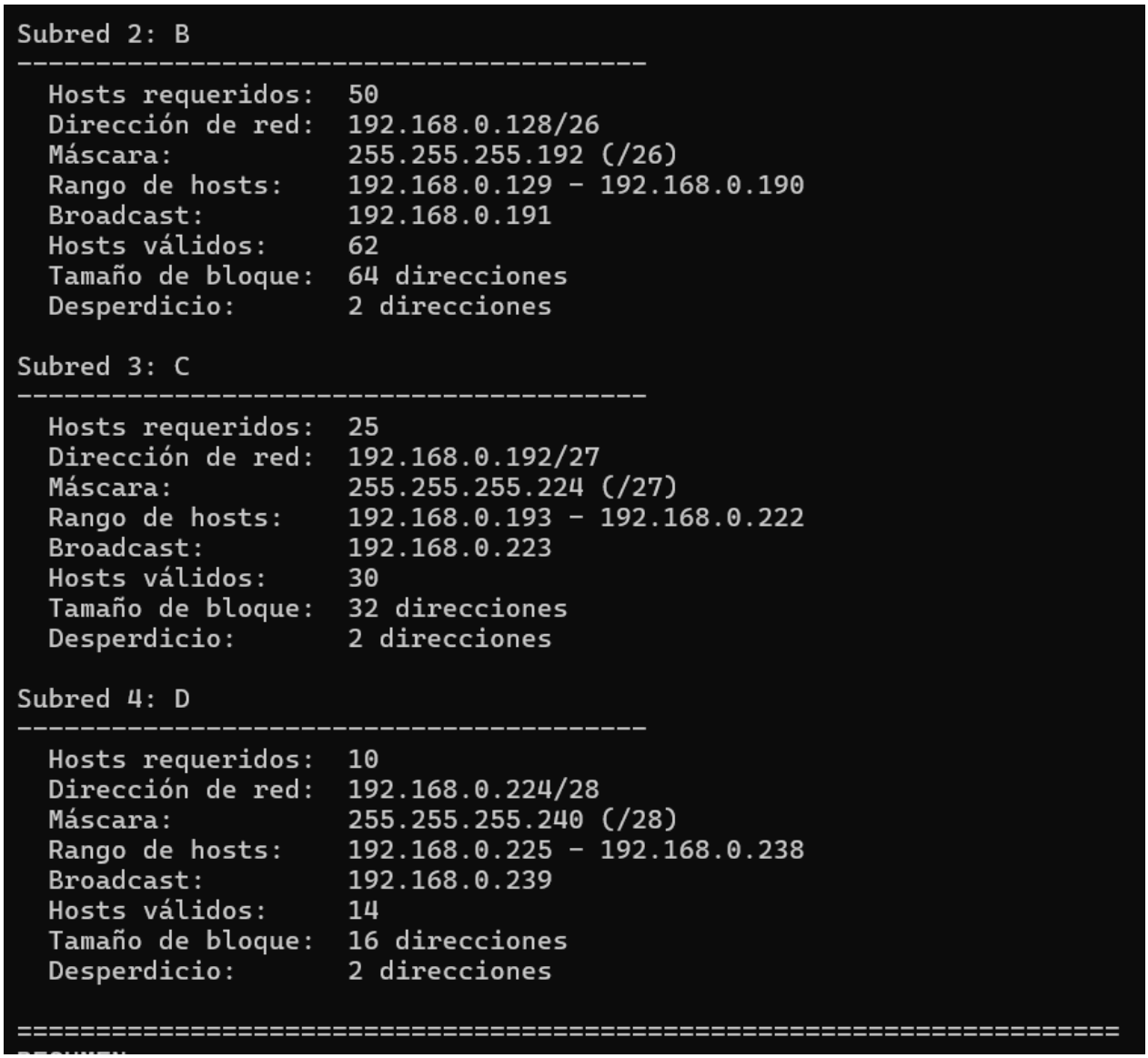


Ilustración 23. Resultados caso 2 p2.

```
RESUMEN
======================================================================

No.  Subred          Red                Hosts Req.   Hosts Disp.  Desperdicio
----------------------------------------------------------------------------
1    A               192.168.0.0/25     100          126          2
2    B               192.168.0.128/26   50           62           2
3    C               192.168.0.192/27   25           30           2
4    D               192.168.0.224/28   10           14           2
----------------------------------------------------------------------------

Total hosts requeridos: 185
Total hosts disponibles: 232
Total desperdicio: 8 direcciones
======================================================================
```

Ilustración 24. Resultados caso 2 p3.

Caso 3 (Limite):

Entrada: 192.168.1.0/30 con 10 hosts

Resultado: Error "Espacio insuficiente"

```
3. CASO LÍMITE VLSM: Red pequeña con muchos hosts
------------------------------------
Error esperado: No hay espacio suficiente para la subred 'S1'. Espacio insuficiente en la red base.

Presione Enter para continuar...
```

Ilustración 25. Resultados caso 3.

CONCLUSIÓN

Blanco Reséndiz Cuauhtémoc

El desarrollo de este proyecto ha proporcionado aprendizajes valiosos en múltiples dimensiones. Técnicamente, hemos profundizado en la comprensión del direccionamiento IP, particularmente en las sutilezas de CIDR y VLSM que a menudo se pasan por alto en tratamientos superficiales del tema. La necesidad de implementar conversiones precisas entre diferentes representaciones de direcciones IP (decimal punteado, binario, entero) nos obligó a entender estos formatos a un nivel fundamental.

Desde la perspectiva de desarrollo de software, aprendimos la importancia del diseño modular y la separación de responsabilidades. La arquitectura elegida, con módulos independientes para utilidades básicas, lógica CIDR y lógica VLSM, demostró su valor cuando necesitamos realizar modificaciones o correcciones, ya que los cambios quedaron confinados a áreas específicas del sistema. Este enfoque también facilitó las pruebas unitarias y la verificación incremental de funcionalidades.