UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO
FACULTAD DE CIENCIAS
REDES DE COMPUTADORAS


# Práctica 4
# Implementación de algoritmos de detección de errores de capa de enlace

**Equipo 4**

**Integrantes :**

**Adrián Aguilera Moreno Num.C:421005200**

**Francisco Contreras Ibarra Num.C: 316083786**

**Aldo Daniel Licona Gómez Num.C: 316263863**

# 1. Implementación de VLSM.

Función que encuentra el número de bits que ocupan los hosts:

```
def encontrar_n(lista_hosts):
    lista_n = []
    for x in lista_hosts:
        lista_n.append(math.ceil(math.log(x, 2)))
    return lista_n
```

Función que encuentra el número de hosts útiles:

```
def encontrar_hosts_utiles(lista_n):
    lista_u = []
    for x in lista_n:
        lista_u.append(pow(2, x) - 2)
    return lista_u
```

Nos indica la máscara en forma de prefijo:

```
def encontrar_mascara(lista_n):
    lista_m = []
    for x in lista_n:
        lista_m.append(32 - x)
    return lista_m
```

Encuentra el valor para el último octeto:

```
def ultimo_octeto(lista_m):
    lista_ultimo_octeto = []
    for x in lista_m:
        sum = 0
        if x > 24 and x <= 32:
            for i in range(7, 7 - (x - 24), -1):
                sum += pow(2, i)
            lista_ultimo_octeto.append(sum)
        elif x > 16:
            for i in range(7, 7 - (x - 16), -1):
                sum += pow(2, i)
            lista_ultimo_octeto.append(sum)
        elif x > 8:
            for i in range(7, 7 - (x - 8), -1):
                sum += pow(2, i)
            lista_ultimo_octeto.append(sum)
        elif x >= 0:
            for i in range(7, 7 - x, -1):
                sum += pow(2, i)
            lista_ultimo_octeto.append(sum)
    return lista_ultimo_octeto
```

Encuentra el número mágico por subred. Estos son los valores del rango válido:

```
def encontrar_magico(lista_ultimo_octeto):
    lista_magicos = []
    for x in lista_ultimo_octeto:
        lista_magicos.append(256 - x)
    return lista_magicos
```

Lectura de nuestra IP por octetos en forma decimal:

```
print("Introduce la direcci n IP por octetos (32 bits):\n")

octeto01 = input("      Escribe el octeto 01 en decimal: ")
octeto02 = input("      Escribe el octeto 02 en decimal: ")
octeto03 = input("      Escribe el octeto 03 en decimal: ")
octeto04 = input("      Escribe el octeto 04 en decimal: ")
prefijo  = input("\n      Ingresa el prefijo asociado a la m scara: ")
```

Devolvemos la IP en su formato standard:

```
print("\n       --> IP: " + octeto01 + "." + octeto02 + "." + octeto03 + "." + octeto04 + "/" +
    prefijo + "\n")
```

Ingresamos la cantidad de subredes a dividir:

```
try:
    hosts = int(input("Introduce la cantidad de subredes deseada en orden decendente: "))
except ValueError:
    print("El n mero de subredes de ser un valor entero positivo, parece que te has equivocado :(")
```

Realizamos lectura y guardamos los valores de hosts en orden decreciente y es de esta manera que se devolverán los datos:

```
lista_hosts = []
contador = 0
while contador < hosts:
    contador += 1
    try:
        temporal = int(input("   Ingresa el n mero de host para la red " + str(contador) + ": "))
    except ValueError:
        print("El n mero valor del host debe ser un valor entero positivo, parece que te has
    equivocado :(")
    lista_hosts.append(temporal)
```

Invocamos cada una de nuestras funciones y giardamos nuestros datos en listas, para usarlas a continuación:

```
lista_n = encontrar_n(lista_hosts)
lista_u = encontrar_hosts_utiles(lista_n)
lista_m = encontrar_mascara(lista_n)
lista_ultimo_octeto = ultimo_octeto(lista_m)
lista_magicos = encontrar_magico(lista_ultimo_octeto)
```

Versión compacta de un toString para nuestros valores de salida:

```
contador = 0
while contador < hosts:
    contador += 1
    representacion = "Subred " + str(contador) + ":\n"
    if contador - 1 != 0:
        representacion += "    ---> IP: " + octeto01 + "." + octeto02 + "." + octeto03 + "." + str(
    lista_ultimo_octeto[contador - 2] + int(octeto04)) + "/" + str(lista_m[contador - 1]) + "\n"
    else:
        representacion += "    ---> IP: " + octeto01 + "." + octeto02 + "." + octeto03 + "." +
    octeto04 + "/" + str(lista_m[contador - 1]) + "\n"
    representacion +=     "          Rango de direcciones: " + str(lista_magicos[contador - 1]) + "
           Broadcast: " + octeto01 + "." + octeto02 + "." + octeto03 + "." + str(lista_ultimo_octeto
    [contador - 1] + int(octeto04) - 1) + "\n"
```

Imprimimos la cadena correspondiente a cada subred:

```
    print(representacion)
```

A continuación se muestra una ejecución de nuestro programa:

```
(base) [aguilera@2806-106e-0013-59b7-6dc1-4c33-ee2f-0118 Práctica07]$ python3 VLSM.py
 ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ VLSM ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

Introduce la dirección IP por octetos (32 bits):

      Escribe el octeto 01 en decimal: 192
      Escribe el octeto 02 en decimal: 168
      Escribe el octeto 03 en decimal: 1
      Escribe el octeto 04 en decimal: 0

      Ingresa el prefijo asociado a la máscara: 24

       --> IP: 192.168.1.0/24

Introduce la cantidad de subredes deseada en orden decendente: 4
   Ingresa el número de host para la red 1: 80
   Ingresa el número de host para la red 2: 50
   Ingresa el número de host para la red 3: 25
   Ingresa el número de host para la red 4: 10
Subred 1:
    ---> IP: 192.168.1.0/25
         Rango de direcciones: 128         Broadcast: 192.168.1.127

Subred 2:
    ---> IP: 192.168.1.128/26
         Rango de direcciones: 64         Broadcast: 192.168.1.191

Subred 3:
    ---> IP: 192.168.1.192/27
         Rango de direcciones: 32         Broadcast: 192.168.1.223

Subred 4:
    ---> IP: 192.168.1.224/28
         Rango de direcciones: 16         Broadcast: 192.168.1.239

(base) [aguilera@2806-106e-0013-59b7-6dc1-4c33-ee2f-0118 Práctica07]$
```

# 2. Concluciones

## 2.1. Adrián

Durante esta práctica análizamos el funcionamiento del algoritmo VLSM con la finalidad de verificar datos importantes para una red de ingreso con su respectivo prefijo asociado a la máscara de red. El programa inscribe una serie de funciones que permiten:

1. Dar el Id de Red por subred en 32 bits con su respectivo prefijo asociado a la submáscara correspondiente.
2. Rango de direcciones útiles.
3. Broadcast.

## 2.2. Francisco

IPV4 es un protocolo de internet el cual está vigente hoy en día, este protocolo permite la comunicación entre máquinas entre sí, donde en esta práctica pudimos saber más sobre IPV4 como su historia, su estructura, clases, entre otros, pero también se vio que y cual es la función de la marca a de red y como hacer el subneto de la máscara de subred el cual es de gran importancia ya que esto nos permite dividir una red en redes en subredes para optimizar las direcciones para cual se vio cada parte que lo compone y el proceso que se requiere realizar para hacer de manera correcta de este procesos con la finalidad de tener los resultados correctos de los componentes de las subredes.

## 2.3. Aldo

En esta práctica vimos un poco más a fondo lo que representan los números que conforman las ip's, es decir, los cuatro octetos con los que cuentan estas, las clases que existen, accesibilidad (que es una característica encontrada de forma frecuente en cuanto a la conectividad a internet) y la perdurabilidad de estas. También se vieron los conceptos de mascara de red, las subredes, el subnetting y una forma de calcular los bits a utilizar, cantidad de host útiles, la máscara de red, entre otros aspectos dada a una dirección IP y elementos relacionados con esta.

# 3. Referencias


- What is VLSM? | NetworkAcademy.io. (s.f.). NetworkAcademy.io. https://www.networkacademy.io/ccna/ip-subnetting/what-is-vlsm
- VLSM CIDR Subnet Calculator. (s.f.). http://www.vlsmcalc.com/