|  |  |  |
| --- | --- | --- |
| ipn | **INSTITUTO POLITÉCNICO NACIONAL**  **ESCUELA SUPERIOR DE CÓMPUTO** |  |

**REDES DE COMPUTADORAS**

**“REPORTE PRÁCTICA 2-SUBNETTING”**

Abstact

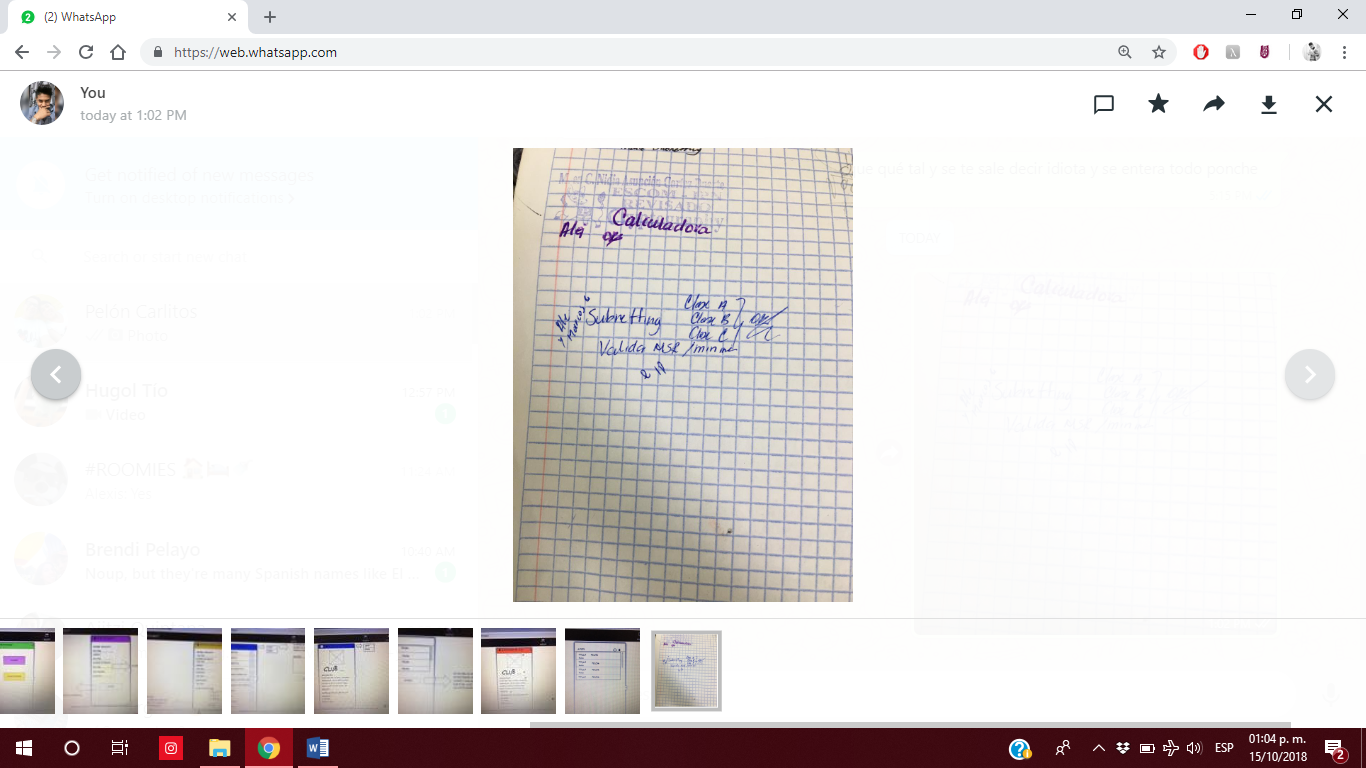
In this report we will see some of characteristics in order to learn about the subnetting, in this report there is a program whose purpose is enter your IP and choose between three options, the first one is by sub network, the second one is by number host and the last one is by IP number and after slash and number of personalized mask, so that you will see on console a table with IP´s sub network, broadcast ad range.

**By:**

**MARCOS OSWALDO VÁZQUEZ MORENO**

**DE LOS SANTOS DÍAZ LUIS ALEJANDRO**

Professor:

MSc. NIDIA ASUNCIÓN CORTEZ DUARTE

OCTOBER 2018

**Índice**

**Contenido**

[Introducción 3](#_Toc526279821)

[Marco Teórico 3](#_Toc526279822)

[Software (librerías, paquetes, herramientas) 3](#_Toc526279823)

[Procedimiento 4](#_Toc526279824)

[Resultado 7](#_Toc526279825)

[Discusión: 12](#_Toc526279826)

[Conclusiones: 13](#_Toc526279827)

[Referencias 13](#_Toc526279828)

[Código 14](#_Toc526279829)

# Introducción

En el siguiente reporte se hablará de la técnica del subnetting que consiste en dada una dirección IP obtener clase de esa IP, obtener el número de bits prestados y dado eso, determinar bits de host, número de host/ subredes totales posibles, obtener una máscara de subred.

# Marco Teórico

Ya desde 1981 Internet funciona tomando como base el llamado Internet Protocol (IP), que consiste en un protocolo de red que regula las vías de transporte de los participantes en las redes. Para enviar un paquete de datos en una red, el emisor debe conocer la dirección IP del receptor.

En la dirección IP se ocultan el Net ID y el Host ID, que permiten la identificación de la red correspondiente y, en ella, del host, que puede ser un PC o una impresora de red. Mediante estos datos, el router tiene la capacidad de transmitir paquetes de datos a los destinatarios correctos.

Los ordenadores solo pueden entender ceros y unos: trabajan así en un sistema numérico binario, motivo por el que las direcciones IP también se construyen siguiendo este principio. El sistema IPv4 utilizado hasta la fecha está compuesto por 32 bits, es decir, por 32 ceros o unos, pero para que esto sea más sencillo y para ahorrar espacio, las direcciones IP se representan siguiendo el formato “dot-decimal notation”, es decir, de forma decimal y divididas por puntos, como indica el ejemplo 192.168.88.3.

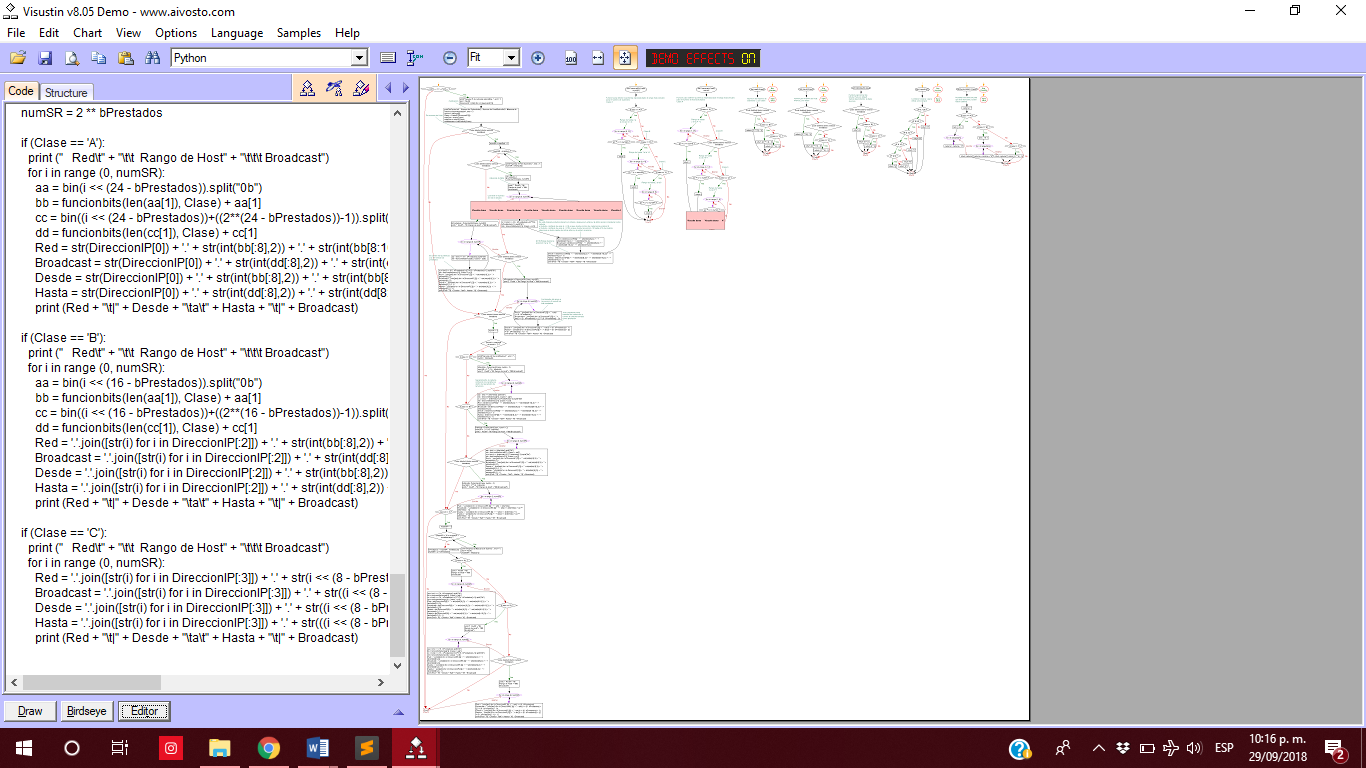
# Software (librerías, paquetes, herramientas)

A continuación, tenemos una lista de las herramientas que se utilizaron para realizar el programa antes mencionado, los paquetes a utilizar y las herramientas que se utilizaron para llevar a cabo dicho programa.

* Editor de texto: Sublime Text 3.
* Compilador: Python-3.6.3.
* Ventana de comandos (Sistema Operativo Windows).
* Lenguaje de alto nivel Python.
* Visustin Editor v8 Demo.

# Procedimiento

Se realizó un programa en lenguaje Python detallado en un diagrama de flujo el cual se puede apreciar en la imagen 1.

  
Imagen 1. Diagrama de flujo.

El código comienza con la declaración de 6 funciones importantes las cuales son:

* **PotenciaSR:** La cual permite obtener la potencia esperada dado el rango más cercano para el número de subredes.
* **PotenciaH:** La cual permite obtener la potencia esperada dado el rango más cercano para el número de host solicitado.
* **maxS:** La cual permite obtener el número de subredes máximo -2 por cada clase.
* **maxH:** La cual permite obtener el número de host por cada clase.
* **bitsDefault:** La cual permite tomar los bits de Red por default dados dependiendo la clase que sea en la validación de clasificación.
* **funcionbits:** La cual es la clave del algoritmo con la que copiamos bits por segmentos y acomodamos de modo que al convertirlo a entero y después a cadena de nuevo podamos obtener los de red menos, los de host y el total de 32.

El programa inicia recibiendo un dato (la dirección IP) ignorando los puntos y utilizándolos como separador con ayuda la función join dependiendo de cada caso y tomando las letras de cada clasificación como operador.

Posteriormente, entra a una opción en la cual se debe elegir por número dependiendo de la necesidad de cada usuario y entra a las validaciones en un IF, teniendo entonces:

1. **Número de Subredes:** Mientras el número de subredes sea mayor al máximo de subredes -2 entonces entra a un if anidado, dependiendo de cada clase que se explicará a continuación.

Para los tres casos se utiliza la función potenciaSR la cual nos devuelve el número de bits prestados. Dependiendo el caso, restamos el número de bits para host menos los bits prestados. Creamos un for de 0 al número de redes ingresadas por el usuario, las cuales serán impresas.

* **Clase A:** De la posición del for hacemos un corrimiento a la izquierda tantos espacios según el resultado de la resta de bits y usamos el método split para meterlos a una lista, transformando el número en binario con “bin()”. De la lista, tomamos la posición donde se haya guardado el número binario y pasamos su longitud y clase a la función funcionbits(), la cual regresa los ceros a la izquierda que resulten de dicho número de red, y los concatenamos con la posición que contenga el número binario para así tener el total de bits de red; guardamos el resultado en una variable. Para el caso de los bits de broadcast, hacemos el mismo corrimiento a la izquierda de los bits resultantes, pero le sumamos 2 elevado al resultado de la resta, menos uno. Llamamos nuevamente a la funcionbits() y concatenamos y volvemos a guardar en una variable.

Posteriormente, procedemos a imprimir los resultados; para la subred, imprimimos directamente el primer octeto, para el segundo, imprimimos la primera variable donde guardamos el resultado, pero, sólo hasta la posición ocho, como está en binario casteamos a int base 2; para el tercero, hacemos lo mismo que el segundo sólo que será de la posición 8 a la 16; para el cuarto nuevamente realizamos el procedimiento pero de la posición 16 a la última. Para el caso del broadcast realizamos el mismo procedimiento, pero con la segunda variable en la que guardamos resultado. Para el rango, para la primera dirección, copiamos lo impreso de la subred, pero le sumamos 1 al último octeto, y para la última dirección, le restamos 1 al último octeto de broadcast

* **Clase B:** El caso es igual al de la clase A, cambiando el número de bits que tenemos para host (en este caso 16) y en la impresión de direcciones, por default imprimimos los 2 primeros octetos y los siguientes nos apoyamos nuevamente de las variables de resultados.
* **Clase C:** Para este caso, las cosas se facilitan aún más, primero disminuyendo a 8 el número de bits para host, lo que mejora el procesamiento de las funciones. Tampoco necesitaremos de las variables de resultados, ya que el corrimiento de bits lo haremos directo en el último octeto.

1. **Número de Host/Subred:** Mientras el número de Host sea mayor al máximo de host o el número de host sea menor entra a otra validación, dependiendo de cada clase que se explicará a continuación.

* **Clase A:** El caso es similar al punto 1, solo que para sacar el número de bits para host de la subred, hay que invocar a la función PotenciaH y elevar 2 al resultado. Hacemos el corrimiento de bits pero ya sin restar nada.
* **Clase B:** Se repite el proceso de la opción uno, pero siguiendo las modificaciones de la clase A de este punto
* **Clase C:** Se repite el proceso de la opción uno, sin embargo, tampoco necesitaremos de las variables de resultados, por lo que los corrimientos se harán directamente en el último octeto

1. **Máscara de Subred personalizada:** Mientras la máscara de subred sea menor a 30 o la máscara de subred sea menor al mínimo de máscara posible, entra los if anidados dependiendo de cada clase que se explicará a continuación.

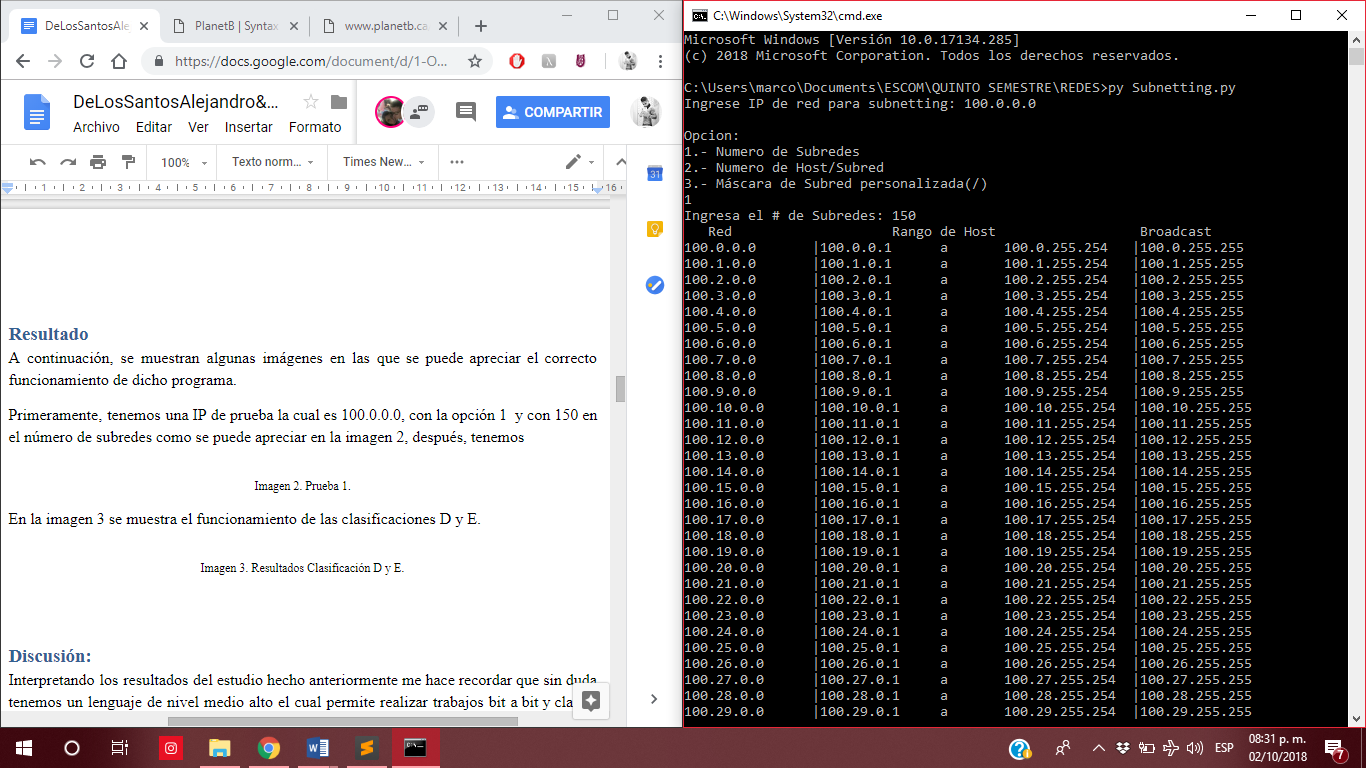
* **Clase A:** Se repite el caso de la opción 1, sin embargo, para los bits prestados, se hace una resta del número de máscara introducido menos los bits de máscara por default resultantes de invocar la función minMascara en la primera parte del programa. Y para el número de subredes, elevamos 2 al resultado de la resta. Nuevamente hacemos un ciclo for, hasta el número de subredes resultantes y realizamos el mismo procedimiento de generación e impresión de octetos de la opción 1
* **Clase B:** Se repite el proceso de la opción 1, siguiendo las modificaciones de la clase A de este punto
* **Clase C:** Se repite el proceso de la opción 1, nuevamente sin variables de resultados, generando los corrimientos de bits dentro del mismo octeto (último).

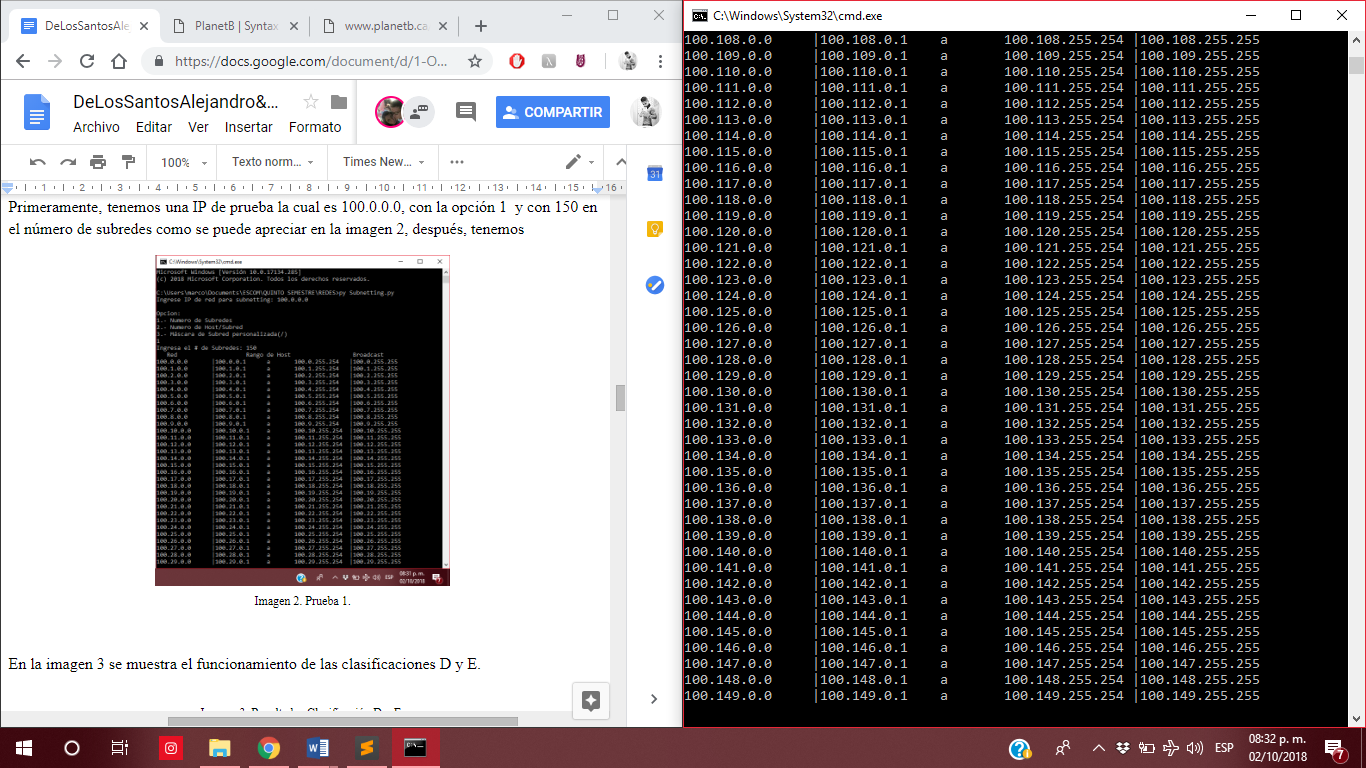
Imprimiendo por último la tabla de lo solicitado, teniendo la IP de subred, el rango y de broadcast de cada una de las redes, además de su máscara de subred.

# Resultado

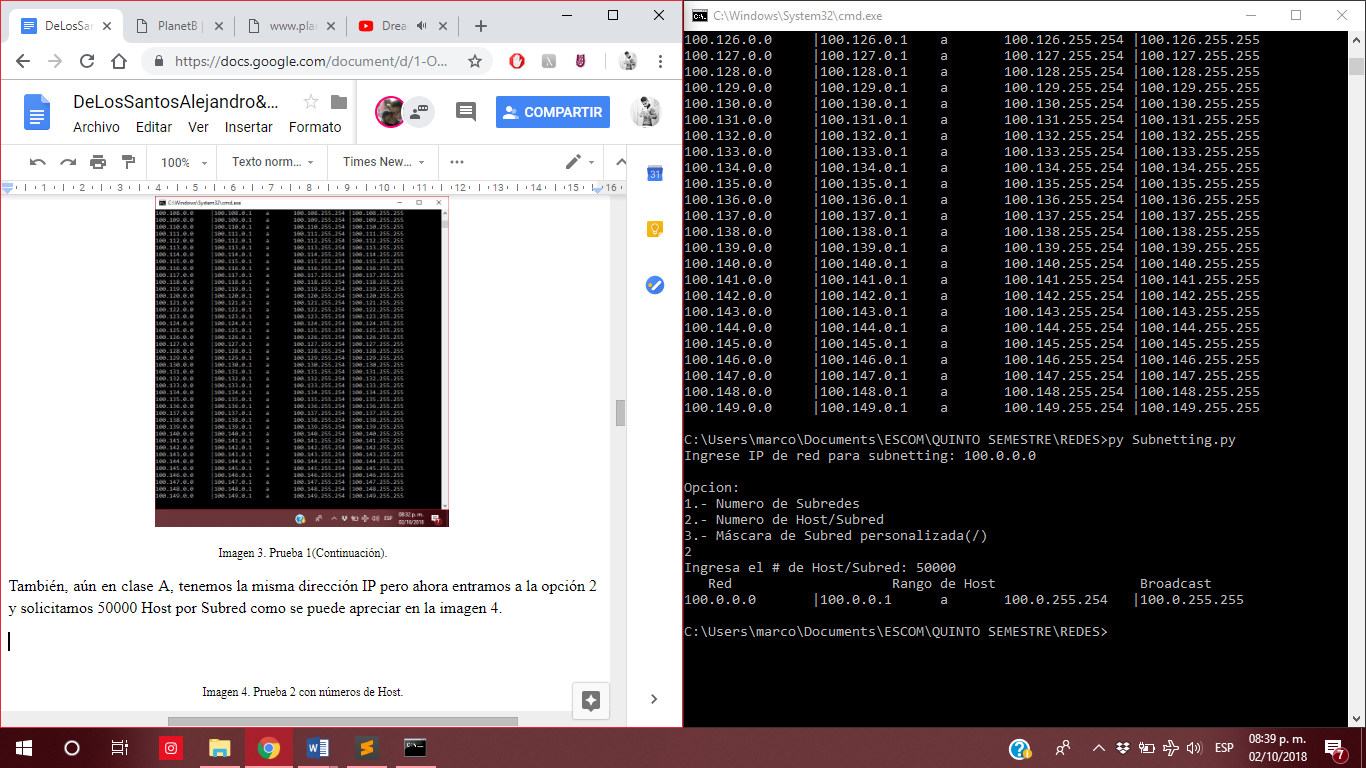
A continuación, se muestran algunas imágenes en las que se puede apreciar el correcto funcionamiento de dicho programa.

Primeramente, tenemos una IP de prueba la cual es 100.0.0.0, con la opción 1 y con 150 en el número de subredes como se puede apreciar en la imagen 2 y 3.

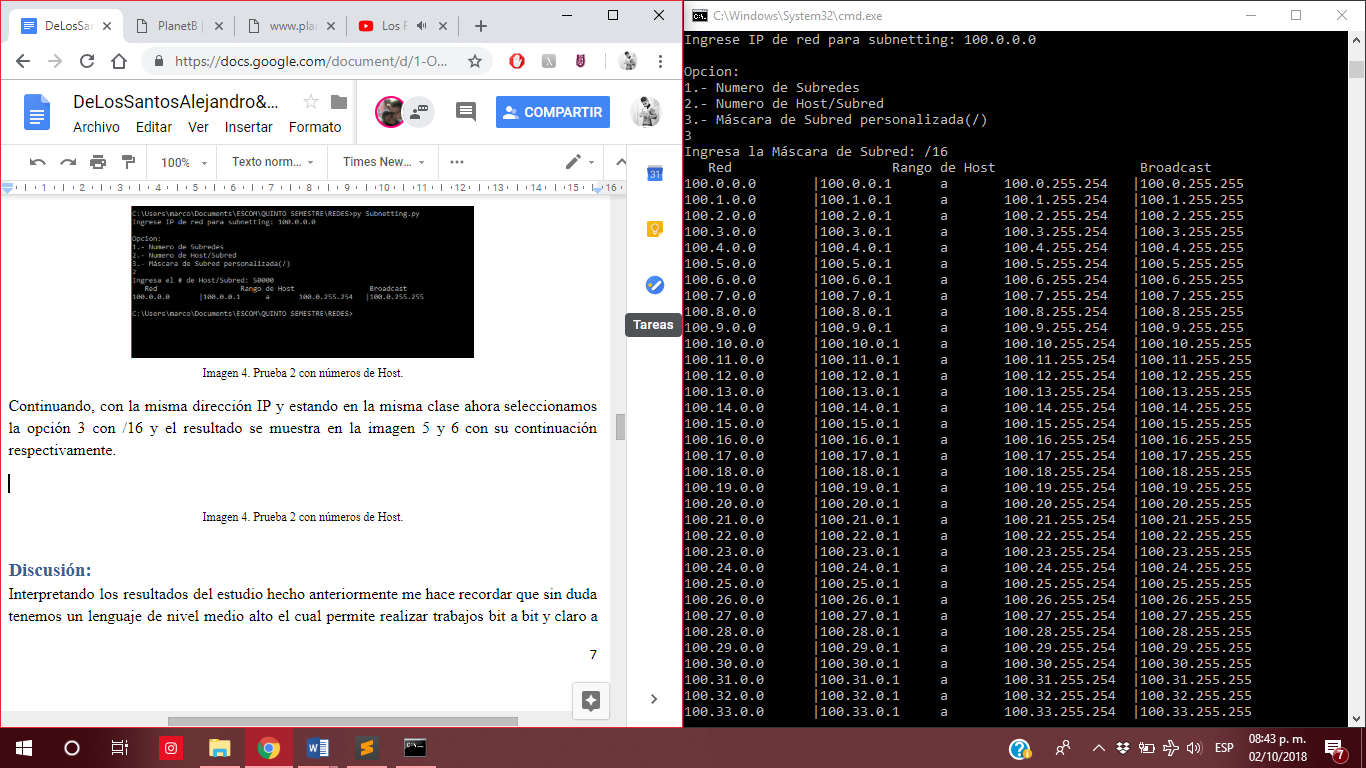
  
Imagen 2. Prueba 1.

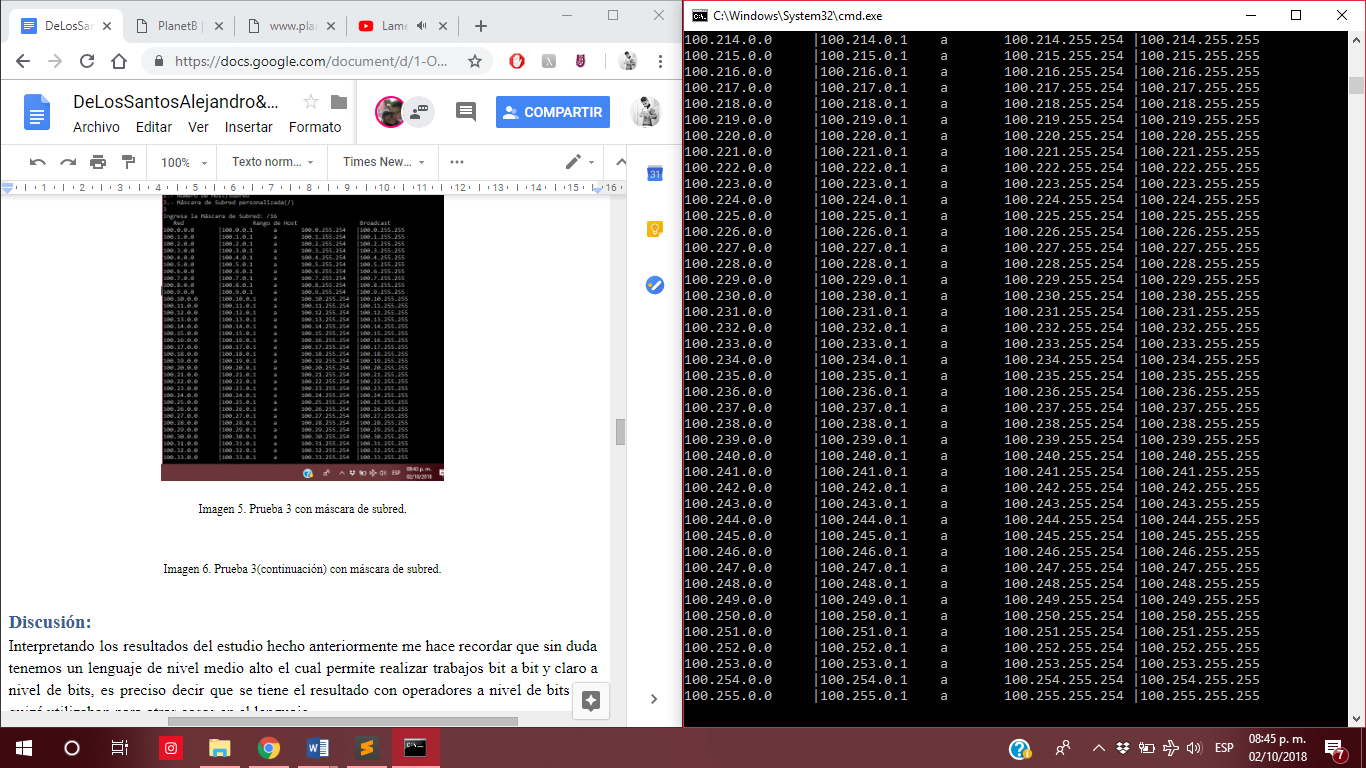
  
Imagen 3. Prueba 1(Continuación).

También, aún en clase A, tenemos la misma dirección IP pero ahora entramos a la opción 2 y solicitamos 50000 Host por Subred como se puede apreciar en la imagen 4.

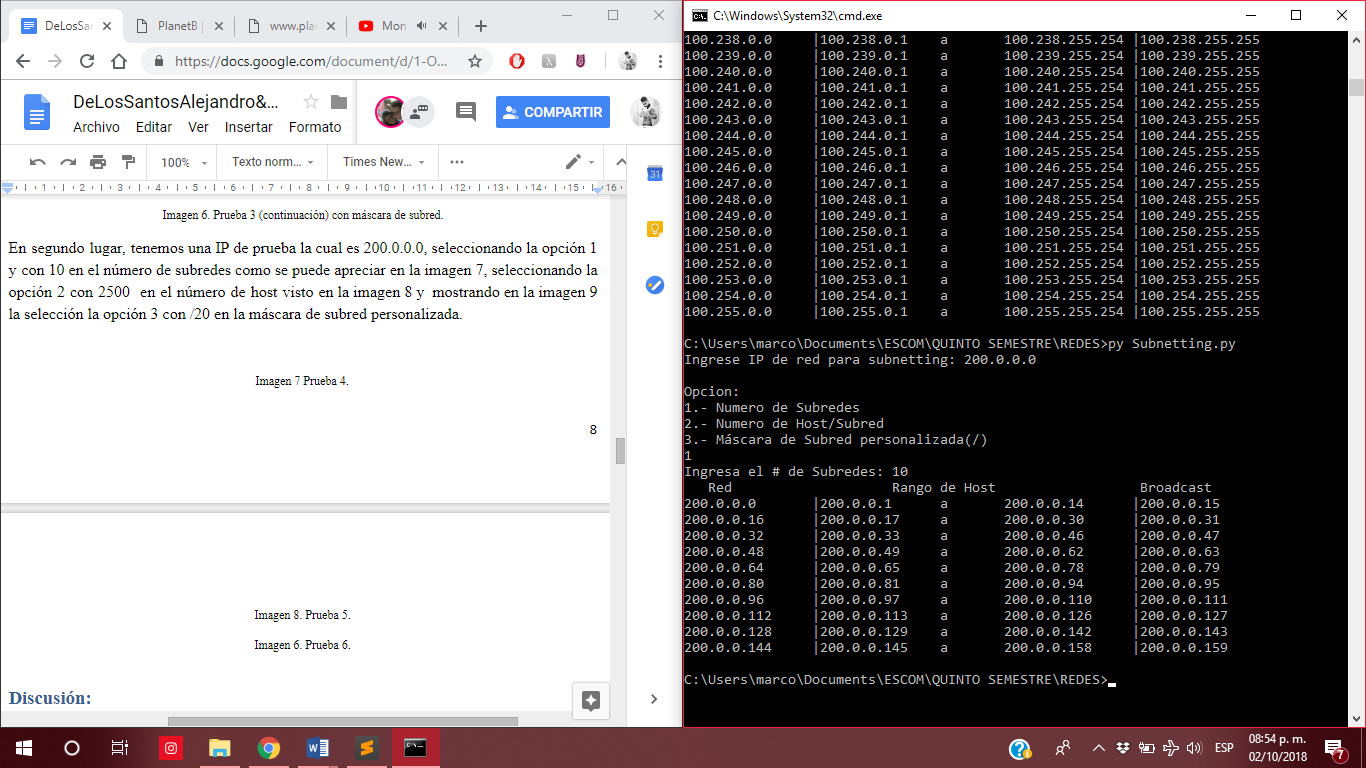
  
Imagen 4. Prueba 2 con números de Host.

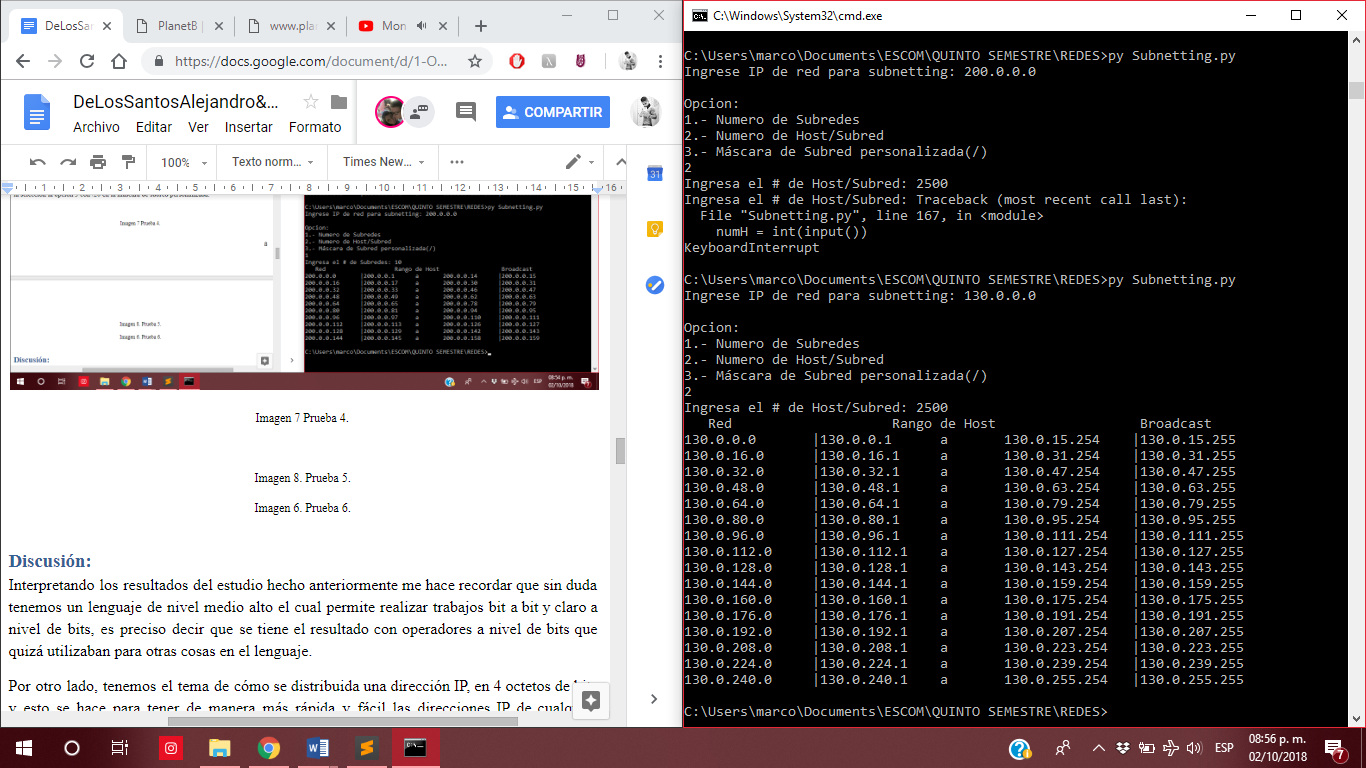
Continuando, con la misma dirección IP y estando en la misma clase ahora seleccionamos la opción 3 con /16 y el resultado se muestra en la imagen 5 y 6 con su continuación respectivamente.

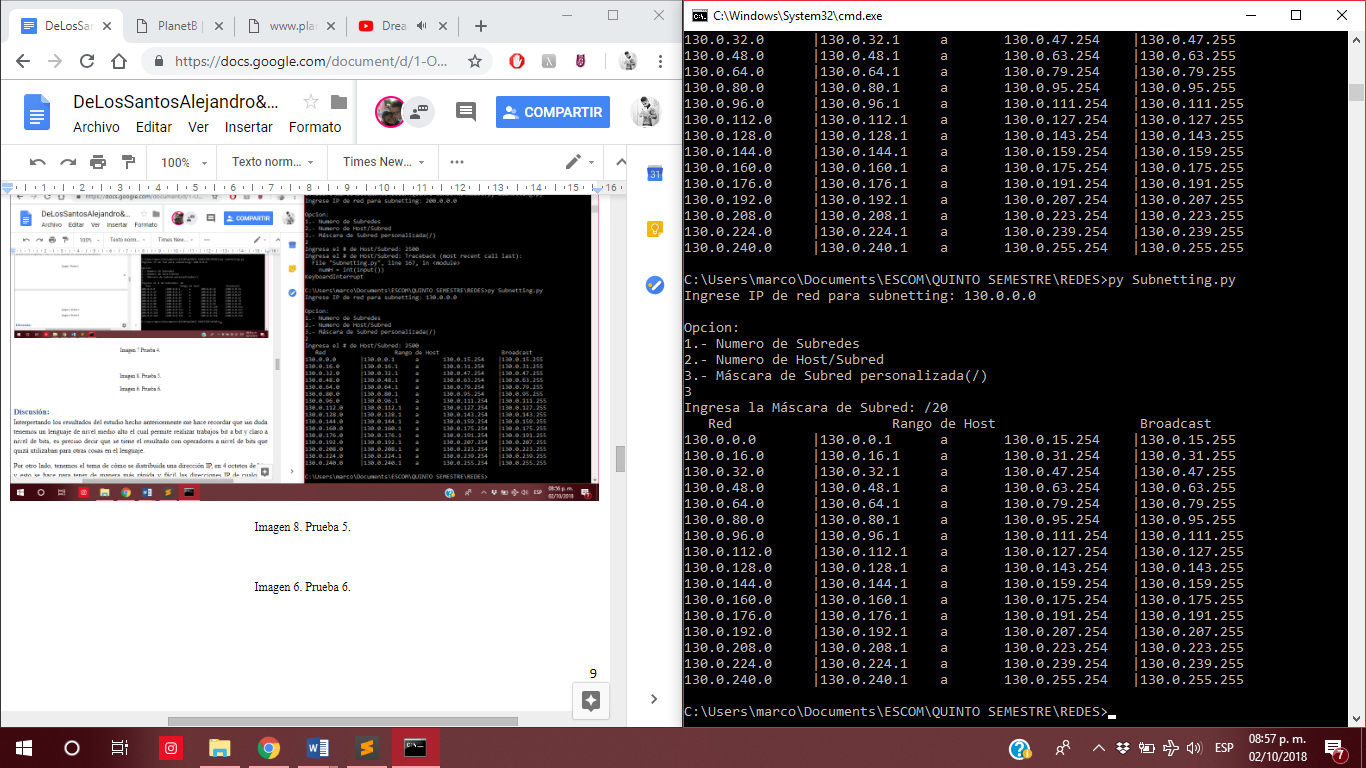
  
Imagen 5. Prueba 3 con máscara de subred.

  
Imagen 6. Prueba 3 (continuación) con máscara de subred.

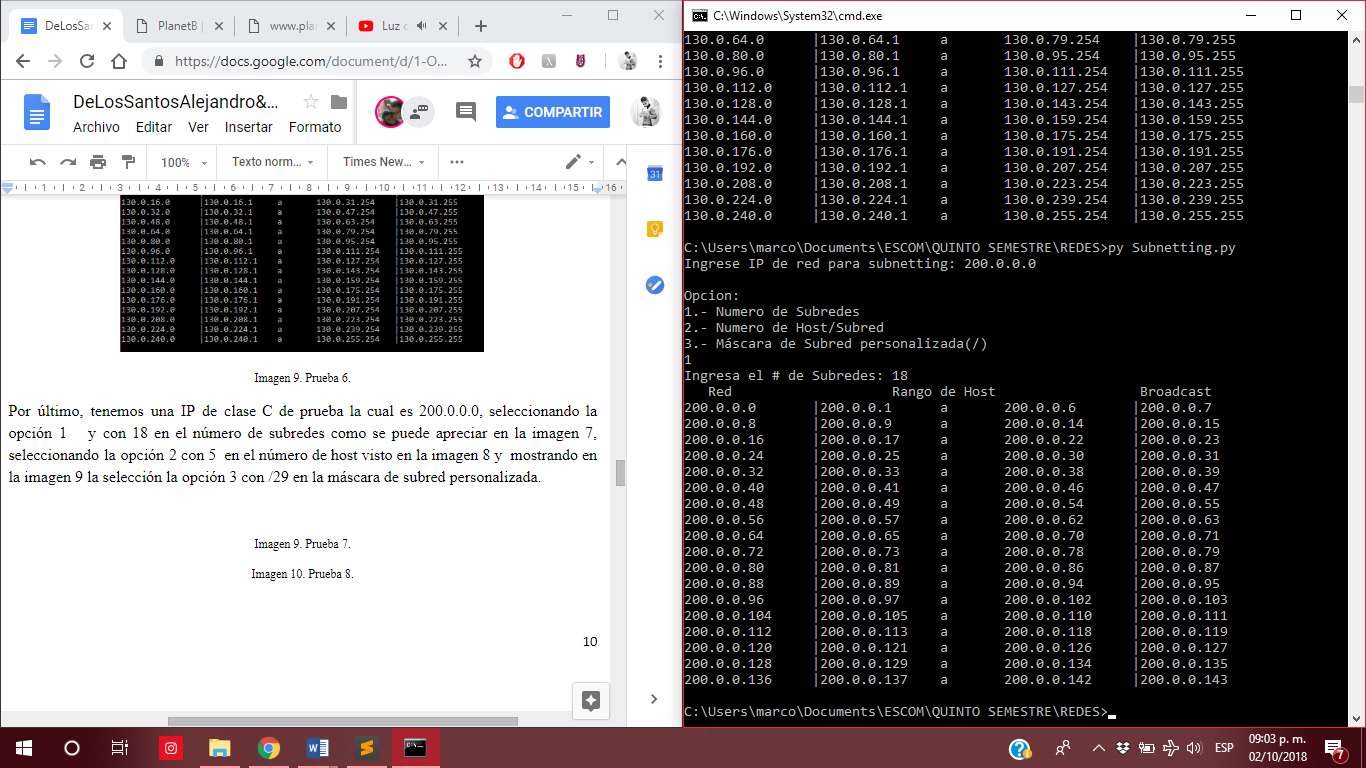
En segundo lugar, tenemos una IP de clase B de prueba la cual es 130.0.0.0, seleccionando la opción 1 y con 10 en el número de subredes como se puede apreciar en la imagen 7, seleccionando la opción 2 con 2500 en el número de host visto en la imagen 8 y mostrando en la imagen 9 la selección la opción 3 con /20 en la máscara de subred personalizada.

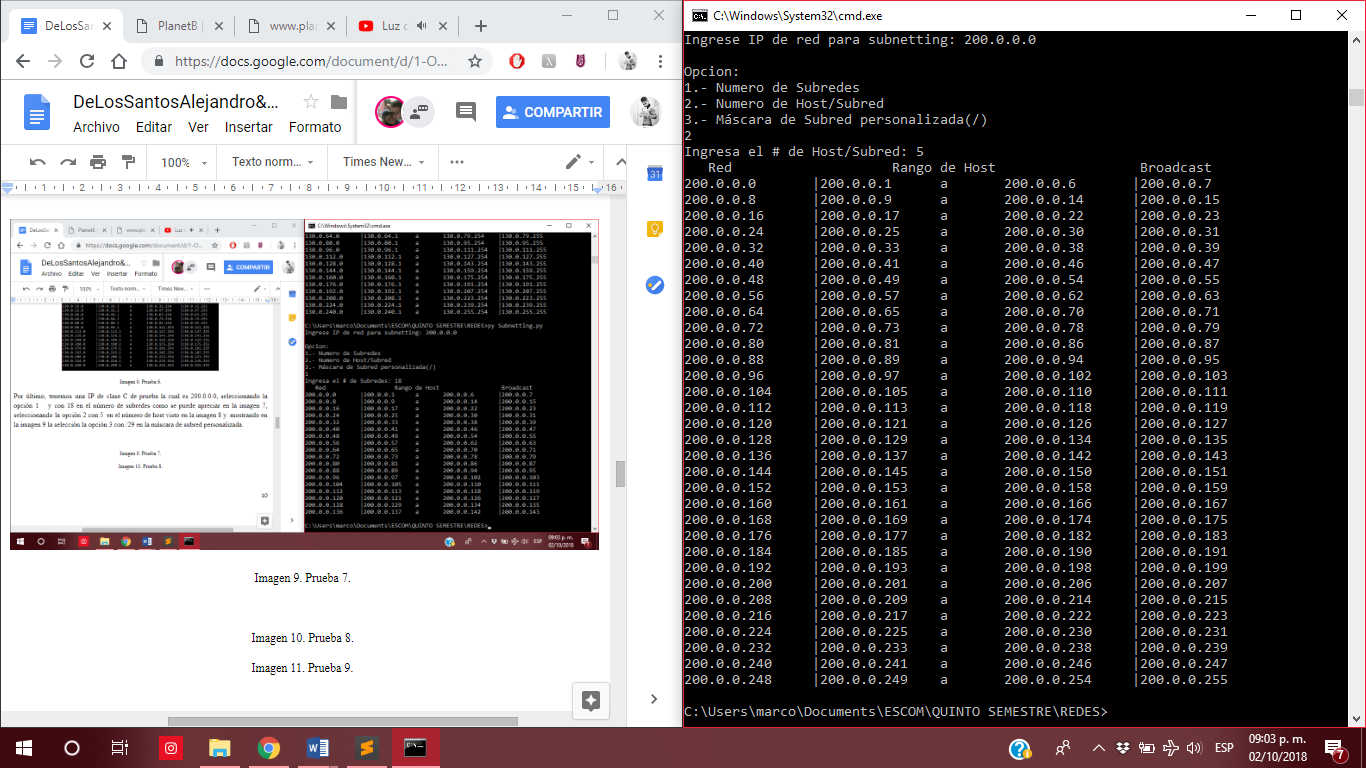
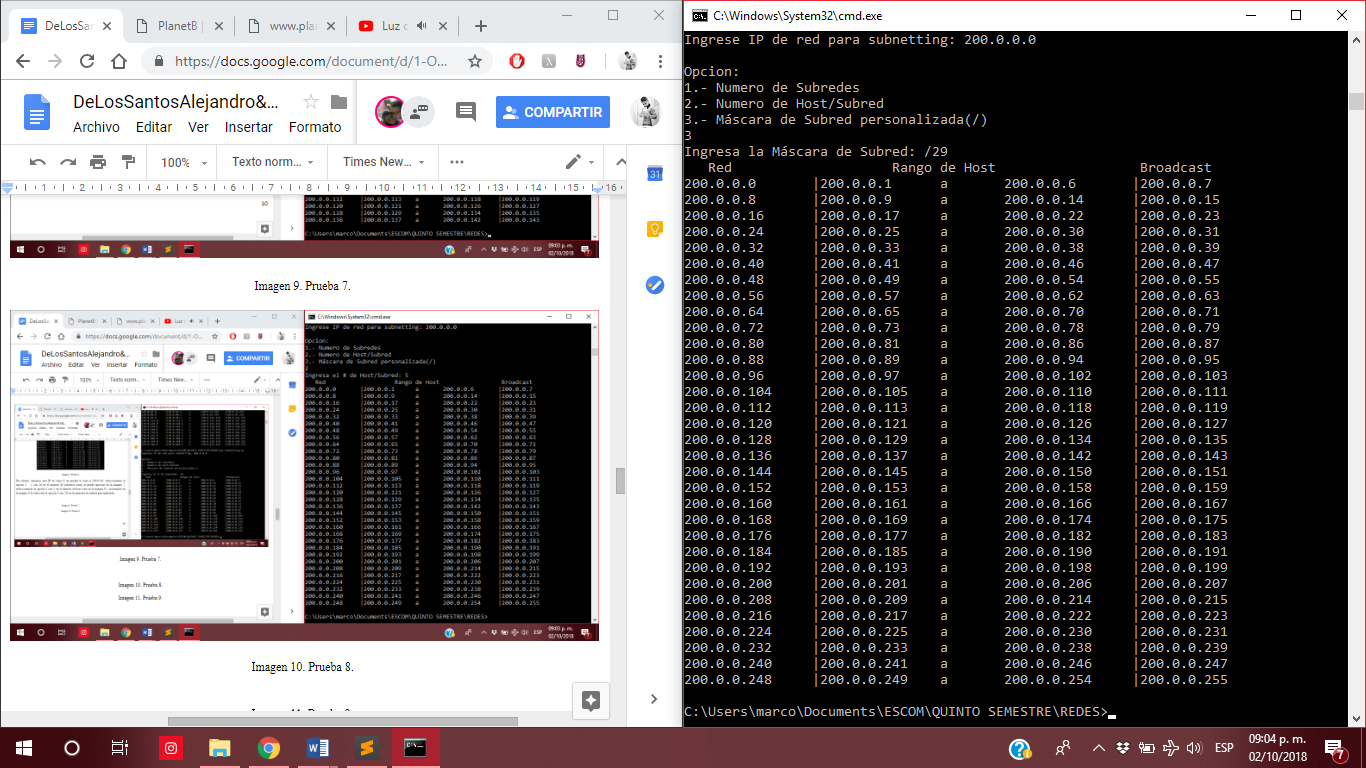
  
Imagen 7 Prueba 4.

  
Imagen 8. Prueba 5.

  
Imagen 9. Prueba 6.

Por último, tenemos una IP de clase C de prueba la cual es 200.0.0.0, seleccionando la opción 1 y con 18 en el número de subredes como se puede apreciar en la imagen 7, seleccionando la opción 2 con 5 en el número de host visto en la imagen 8 y mostrando en la imagen 9 la selección la opción 3 con /29 en la máscara de subred personalizada.

  
Imagen 9. Prueba 7.

  
Imagen 10. Prueba 8.  
  
Imagen 11. Prueba 9.

# Discusión:

Interpretando los resultados del estudio hecho anteriormente me hace recordar que sin duda tenemos un lenguaje de nivel medio alto el cual permite realizar trabajos bit a bit y claro a nivel de bits, es preciso decir que se tiene el resultado con operadores a nivel de bits que quizá utilizaban para otras cosas en el lenguaje.

Por otro lado, tenemos el tema de cómo se distribuida una dirección IP, en 4 octetos de bits y esto se hace para tener de manera más rápida y fácil las direcciones IP de cualquier dispositivo con tarjeta de red. Las direcciones IP están distribuidas por regiones y por formas de acceso en la red, datos móviles y distintos distribuidores de servicios de internet.

Dicho lo anterior, es importante mencionar que la teoría vista en clase se puede llevar a cabo en la lógica de la programación, así como los que inventaron la forma de conectividad en red de esta manera lo lograron conseguir.

Aun cuando no es mi fortaleza la programación en este lenguaje logré llevarlo a cabo, claro que no es el más difícil, pero me gustó mucho como a nivel de bits lo logré hacer y sin ningún problema.

# Conclusiones:

Para el desarrollo de esta práctica fue muy importante tener sumamente claro el concepto y práctica en lápiz y papel de lo que es el subnetting, ya que había que transportar esta teoría a la codificación.

Aprendimos a usar mejor las funciones que algunos lenguajes de programación como python ofrecen.

También aprendimos que para el subnetting un mismo algoritmo (estrictamente para codificar), sirve para resolver casos más básicos del mismo.

# Referencias

|  |  |
| --- | --- |
| [1] | N. Córtez, «Explicación del problema,» CDMX, 2018. |
| [2] | U. I. d. Valencia, «https://www.universidadviu.com/funciona-protocolo-ip/,» [En línea]. Available: https://www.universidadviu.com/funciona-protocolo-ip/. |
| [3] | B. O. Moncada, Programando en C a bajo nivel, Buenos Aires: Facultad de Ingeniería, 2011. |
| [4] | Google, «Syntax Highlight Code In Word Documents,» Google, 2012. [En línea]. Available: http://www.planetb.ca/syntax-highlight-word. [Último acceso: 22 Septiembre 2018]. |

# Código

El código que se muestra en seguida fue el implementado para el programa llamado Subnetting. [4]

* """
* Escuela Superior de Computo
* Entrega: 24 de Septimebre de 2018
* Redes de Computadoras
* Alumnos:
* De los Santos Díaz Luis Alejandro 2017630451
* Vázquez Moreno Marcos Oswaldo 2016601777
* Programa que reraliza el Subnetting con una IP dada, entrando a la opcion que puede ser
* por numero de subredes deseadas
* por numero de host deseados
* por mascara personalizada con /#
* Realizado todo lo anterior mediante una validacion de clasificacion de red
* """
* **def** PotenciaSR(Clase, numSR): #Funcion para obtener la potencia esperada dado el rango mas cercano para el numero de subredes
* **if** (Clase == 'A'): #Clase A
* **for** i **in** range (0, 22): #Rango de hasta 2 a la 22 (por el -2)
* **if** ((2 \*\* i) >= numSR):
* **return** i
* **if** (Clase == 'B'): #Clase B
* **for** i **in** range (0, 14): #Rango de hasta 2 a la 14
* **if** ((2 \*\* i) >= numSR):
* **return** i
* **if** (Clase == 'C'): #Clase C
* **for** i **in** range (0, 6): #Rango de hasta 2 a la 6
* **if** ((2 \*\* i) >= numSR):
* **return** i
* **def** PotenciaH(Clase, numH): #Funcion para obtener la potencia esperada dado el rango mas cercano para el numero de host solicitados
* **if** (Clase == 'A'): #Clase A
* **for** i **in** range (1, 25): #Rango de hasta 2 a la 24 +2
* **if** ((2 \*\* i) >= numH):
* **return** i
* **if** (Clase == 'B'): #Clase B
* **for** i **in** range (1, 17): #Rango de hasta 2 a la 16 +1
* **if** ((2 \*\* i) >= numH):
* **return** i
* **if** (Clase == 'C'): #Clase C
* **for** i **in** range (1, 9): #Rango de hasta 2 a la 8 +1
* **if** ((2 \*\* i) >= numH):
* **return** i
* **def** maxS(Clase): # Numeor del maximo de Subredes -2 por clase
* **if** (Clase == 'A'):
* **return** ((2 \*\* 22) - 2)
* **if** (Clase == 'B'):
* **return** ((2 \*\* 14) - 2)
* **if** (Clase == 'C'):
* **return** ((2 \*\* 6) - 2)
* **def** maxH(Clase): #Numero del maximo de host menos 2 por clase
* **if** (Clase == 'A'):
* **return** ((2 \*\* 24) - 2)
* **if** (Clase == 'B'):
* **return** ((2 \*\* 16) - 2)
* **if** (Clase == 'C'):
* **return** ((2 \*\* 8) - 2)
* **def** bitsDefault(Clase): #Funcion para tomar los bits de Red por default dados dependiendo la clase que sea
* **if** (Clase == 'A'):
* **return** 8
* **if** (Clase == 'B'):
* **return** (8 \* 2)
* **if** (Clase == 'C'):
* **return** (8 \* 3)
* **def** ClaseIP(IP): #Condicion clasica :v para entrar a los rangos
* **if** (IP & 128):
* **if** (IP & 64):
* **if** (IP & 32):
* **if** (IP & 16):
* **if** (IP & 8):
* **return** "E"
* **else**:
* **return** "E"
* **else**:
* **return** "D"
* **else**:
* **return** "C"
* **else**:
* **return** "B"
* **else**:
* **return** "A"
* **def** funcionbits(t, Clase): #Nuestra bella funcion para que todo funcione y poder hacer casteos
* cadena = ''
* **if** (Clase == 'A'):
* tam = 24 - t
* **for** i **in** range(tam):
* cadena = cadena + "0"
* **return** cadena
* **if** (Clase == 'B'):
* tam = 16 - t
* **for** i **in** range(tam):
* cadena = cadena + "0"
* **return** cadena
* **if** (Clase == 'C'):
* **return** ((2 \*\* 6) - 2)

* **if** \_\_name\_\_ == '\_\_main\_\_':
* **print**("Ingrese IP de red para subnetting: ", end = '')
* aux = input()
* DireccionIP = [int(i) **for** i **in** ((aux).split('.'))]
* **print**("\nOpcion:\n1.- Numero de Subredes\n2.- Numero de Host/Subred\n3.- Máscara de Subred personalizada(/)\n", end = '') #multiopcion
* opcion = int(input())
* Clase = ClaseIP(DireccionIP[0])
* maxSub = maxS(Clase)
* maxHost = maxH(Clase)
* minMascara = bitsDefault(Clase)
* #Por numero de Host
* **if** (opcion == 1):
* numSR = maxSub + 1
* **while** (numSR > maxSub):
* **print**("Ingresa el # de Subredes: ", end = '')
* numSR = int(input())
* **if** (Clase == 'A'):
* bPrestados = PotenciaSR(Clase, numSR)
* **print** (" Red\t" + "\t\t Rango de Host" + "\t\t\t Broadcast") #cabecera de tabla
* **for** i **in** range (0, numSR):
* aa = bin(i << (24 - bPrestados)).split("0b") #Convierte el numero de red en binario
* bb = funcionbits(len(aa[1]), Clase) + aa[1]
* cc = bin((i << (24 - bPrestados))+((2\*\*(24 - bPrestados))-1)).split("0b") #Guarda los bits para el numeor de broacast
* dd = funcionbits(len(cc[1]), Clase) + cc[1]
* #Lo que hacemos aquí es utilizar una concatenación de los 8 bits de cada octeto dependiendo la clase,
* #De esta manera podemos tenerlo en binario, despues en entero y al utlimo poder concatenar como cadena,
* # el metodo o sintaxis de usar el -> [:8] es que desde el inicio de cadena a la posicion 8
* # el metodo o sintaxis de usar el -> [16:] es que desde la posicion 16 hasta el fin de cadena
* Red = str(DireccionIP[0]) + '.' + str(int(bb[:8],2)) + '.' + str(int(bb[8:16],2)) + '.' + str(int(bb[16:],2)) #Hacemos un doble casteo de bits a entero y de entero a cadena
* Broadcast = str(DireccionIP[0]) + '.' + str(int(dd[:8],2)) + '.' + str(int(dd[8:16],2)) + '.' + str(int(dd[16:],2))
* # [8:16] Rango desde la posicion 8 a la 16
* Desde = str(DireccionIP[0]) + '.' + str(int(bb[:8],2)) + '.' + str(int(bb[8:16],2)) + '.' + str(int(bb[16:],2) + 1)
* Hasta = str(DireccionIP[0]) + '.' + str(int(dd[:8],2)) + '.' + str(int(dd[8:16],2)) + '.' + str(int(dd[16:],2) - 1)
* **print** (Red + "\t|" + Desde + "\ta\t" + Hasta + "\t|" + Broadcast)
* **if** (Clase == 'B'):
* bPrestados = PotenciaSR(Clase, numSR)
* **print** (" Red\t" + "\t\t Rango de Host" + "\t\t\t Broadcast")
* **for** i **in** range (0, numSR):
* aa = bin(i << (16 - bPrestados)).split("0b")
* bb = funcionbits(len(aa[1]), Clase) + aa[1]
* cc = bin((i << (16 - bPrestados))+((2\*\*(16 - bPrestados))-1)).split("0b") #El cambio es la potencia a la 16 menos los prestados
* dd = funcionbits(len(cc[1]), Clase) + cc[1]
* Red = '.'.join([str(i) **for** i **in** DireccionIP[:2]]) + '.' + str(int(bb[:8],2)) + '.' + str(int(bb[8:],2))
* Broadcast = '.'.join([str(i) **for** i **in** DireccionIP[:2]]) + '.' + str(int(dd[:8],2)) + '.' + str(int(dd[8:],2))
* Desde = '.'.join([str(i) **for** i **in** DireccionIP[:2]]) + '.' + str(int(bb[:8],2)) + '.' + str(int(bb[8:],2) + 1)
* Hasta = '.'.join([str(i) **for** i **in** DireccionIP[:2]]) + '.' + str(int(dd[:8],2)) + '.' + str(int(dd[8:],2) - 1)
* **print** (Red + "\t|" + Desde + "\ta\t" + Hasta + "\t|" + Broadcast)
* **if** (Clase == 'C'):
* bPrestados = PotenciaSR(Clase, numSR)
* **print** (" Red\t" + "\t\t Rango de Host" + "\t\t\t Broadcast")
* **for** i **in** range (0, numSR):
* Red = '.'.join([str(i) **for** i **in** DireccionIP[:3]]) + '.' + str(i << (8 - bPrestados)) #Corrimiento del rango a la posicion 8 menos los bits prestados
* Broadcast = '.'.join([str(i) **for** i **in** DireccionIP[:3]]) + '.' + str((i << (8 - bPrestados)) + ((2 \*\* (8 - bPrestados)) - 1))
* Desde = '.'.join([str(i) **for** i **in** DireccionIP[:3]]) + '.' + str((i << (8 - bPrestados)) + 1) #Join concatena cierta cantidad de cadena de a cuerdo al caracter tomado como apuntador
* Hasta = '.'.join([str(i) **for** i **in** DireccionIP[:3]]) + '.' + str(((i << (8 - bPrestados)) + ((2 \*\* (8 - bPrestados)) - 1)) - 1)
* **print** (Red + "\t|" + Desde + "\ta\t" + Hasta + "\t|" + Broadcast)
* **if** (opcion == 2):
* numH = -1
* **while** (numH > maxHost **or** numH < 2):
* **print**("Ingresa el # de Host/Subred: ", end = '')
* numH = int(input())
* **if** (Clase == 'A'):
* bitsHots = PotenciaH(Clase, numH + 2)
* numSR = 2 \*\* (16 - bitsHots)
* **print** (" Red\t" + "\t\t Rango de Host" + "\t\t\t Broadcast")
* **for** i **in** range (0, numSR):
* aa = bin(i << (bitsHots)).split("0b") #Separamiento de cadena omitiendo los carácteres dentro del parametro de la funcion
* bb = funcionbits(len(aa[1]), Clase) + aa[1]
* cc = bin((i << (bitsHots))+((2\*\*(bitsHots))-1)).split("0b")
* dd = funcionbits(len(cc[1]), Clase) + cc[1]
* Red = str(DireccionIP[0]) + '.' + str(int(bb[:8],2)) + '.' + str(int(bb[8:16],2)) + '.' + str(int(bb[16:],2))
* Broadcast = str(DireccionIP[0]) + '.' + str(int(dd[:8],2)) + '.' + str(int(dd[8:16],2)) + '.' + str(int(dd[16:],2))
* Desde = str(DireccionIP[0]) + '.' + str(int(bb[:8],2)) + '.' + str(int(bb[8:16],2)) + '.' + str(int(bb[16:],2) + 1)
* Hasta = str(DireccionIP[0]) + '.' + str(int(dd[:8],2)) + '.' + str(int(dd[8:16],2)) + '.' + str(int(dd[16:],2) - 1)
* **print** (Red + "\t|" + Desde + "\ta\t" + Hasta + "\t|" + Broadcast)

* **if** (Clase == 'B'):
* bitsHots = PotenciaH(Clase, numH + 2)
* numSR = 2 \*\* (16 - bitsHots)
* **print** (" Red\t" + "\t\t Rango de Host" + "\t\t\t Broadcast")
* **for** i **in** range (0, numSR):
* aa = bin(i << (bitsHots)).split("0b")
* bb = funcionbits(len(aa[1]), Clase) + aa[1]
* cc = bin((i << (bitsHots))+((2\*\*(bitsHots))-1)).split("0b")
* dd = funcionbits(len(cc[1]), Clase) + cc[1]
* Red = '.'.join([str(i) **for** i **in** DireccionIP[:2]]) + '.' + str(int(bb[:8],2)) + '.' + str(int(bb[8:],2))
* Broadcast = '.'.join([str(i) **for** i **in** DireccionIP[:2]]) + '.' + str(int(dd[:8],2)) + '.' + str(int(dd[8:],2))
* Desde = '.'.join([str(i) **for** i **in** DireccionIP[:2]]) + '.' + str(int(bb[:8],2)) + '.' + str(int(bb[8:],2) + 1)
* Hasta = '.'.join([str(i) **for** i **in** DireccionIP[:2]]) + '.' + str(int(dd[:8],2)) + '.' + str(int(dd[8:],2) - 1)
* **print** (Red + "\t|" + Desde + "\ta\t" + Hasta + "\t|" + Broadcast)
* **if** (Clase == 'C'):
* bitsHots= PotenciaH(Clase, numH + 2)
* numSR = 2 \*\* (8 - bitsHots)
* **print** (" Red\t" + "\t\t Rango de Host" + "\t\t\t Broadcast")
* **for** i **in** range (0, numSR):
* Red = '.'.join([str(i) **for** i **in** DireccionIP[:3]]) + '.' + str(i << bitsHots)
* Broadcast = '.'.join([str(i) **for** i **in** DireccionIP[:3]]) + '.' + str((i << (bitsHots)) + ((2 \*\* (bitsHots)) - 1))
* Desde = '.'.join([str(i) **for** i **in** DireccionIP[:3]]) + '.' + str((i << (bitsHots)) + 1)
* Hasta = '.'.join([str(i) **for** i **in** DireccionIP[:3]]) + '.' + str(((i << (bitsHots)) + ((2 \*\* (bitsHots)) - 1)) - 1)
* **print** (Red + "\t|" + Desde + "\ta\t" + Hasta + "\t|" + Broadcast)
* **if** (opcion == 3):
* maskSR = -1
* **while** (maskSR > 30 **or** maskSR < minMascara):
* **print** ("Ingresa la Máscara de Subred: ", end = '')
* aux = input()
* maskSR = int(aux[1:])
* bPrestados = maskSR - minMascara
* numSR = 2 \*\* bPrestados
* **if** (Clase == 'A'):
* **print** (" Red\t" + "\t\t Rango de Host" + "\t\t\t Broadcast")
* **for** i **in** range (0, numSR):
* aa = bin(i << (24 - bPrestados)).split("0b")
* bb = funcionbits(len(aa[1]), Clase) + aa[1]
* cc = bin((i << (24 - bPrestados))+((2\*\*(24 - bPrestados))-1)).split("0b")
* dd = funcionbits(len(cc[1]), Clase) + cc[1]
* Red = str(DireccionIP[0]) + '.' + str(int(bb[:8],2)) + '.' + str(int(bb[8:16],2)) + '.' + str(int(bb[16:],2))
* Broadcast = str(DireccionIP[0]) + '.' + str(int(dd[:8],2)) + '.' + str(int(dd[8:16],2)) + '.' + str(int(dd[16:],2))
* Desde = str(DireccionIP[0]) + '.' + str(int(bb[:8],2)) + '.' + str(int(bb[8:16],2)) + '.' + str(int(bb[16:],2) + 1)
* Hasta = str(DireccionIP[0]) + '.' + str(int(dd[:8],2)) + '.' + str(int(dd[8:16],2)) + '.' + str(int(dd[16:],2) - 1)
* **print** (Red + "\t|" + Desde + "\ta\t" + Hasta + "\t|" + Broadcast)
* **if** (Clase == 'B'):
* **print** (" Red\t" + "\t\t Rango de Host" + "\t\t\t Broadcast")
* **for** i **in** range (0, numSR):
* aa = bin(i << (16 - bPrestados)).split("0b")
* bb = funcionbits(len(aa[1]), Clase) + aa[1]
* cc = bin((i << (16 - bPrestados))+((2\*\*(16 - bPrestados))-1)).split("0b")
* dd = funcionbits(len(cc[1]), Clase) + cc[1]
* Red = '.'.join([str(i) **for** i **in** DireccionIP[:2]]) + '.' + str(int(bb[:8],2)) + '.' + str(int(bb[8:],2))
* Broadcast = '.'.join([str(i) **for** i **in** DireccionIP[:2]]) + '.' + str(int(dd[:8],2)) + '.' + str(int(dd[8:],2))
* Desde = '.'.join([str(i) **for** i **in** DireccionIP[:2]]) + '.' + str(int(bb[:8],2)) + '.' + str(int(bb[8:],2) + 1)
* Hasta = '.'.join([str(i) **for** i **in** DireccionIP[:2]]) + '.' + str(int(dd[:8],2)) + '.' + str(int(dd[8:],2) - 1)
* **print** (Red + "\t|" + Desde + "\ta\t" + Hasta + "\t|" + Broadcast)
* **if** (Clase == 'C'):
* **print** (" Red\t" + "\t\t Rango de Host" + "\t\t\t Broadcast")
* **for** i **in** range (0, numSR):
* Red = '.'.join([str(i) **for** i **in** DireccionIP[:3]]) + '.' + str(i << (8 - bPrestados))
* Broadcast = '.'.join([str(i) **for** i **in** DireccionIP[:3]]) + '.' + str((i << (8 - bPrestados)) + ((2 \*\* (8 - bPrestados)) - 1))
* Desde = '.'.join([str(i) **for** i **in** DireccionIP[:3]]) + '.' + str((i << (8 - bPrestados)) + 1)
* Hasta = '.'.join([str(i) **for** i **in** DireccionIP[:3]]) + '.' + str(((i << (8 - bPrestados)) + ((2 \*\* (8 - bPrestados)) - 1)) - 1)
* **print** (Red + "\t|" + Desde + "\ta\t" + Hasta + "\t|" + Broadcast)