INSTITUTO POLITÉCNICO NACIONAL
ESCUELA SUPERIOR DE CÓMPUTO

Computer Networks

"Subnetting"

Abstract

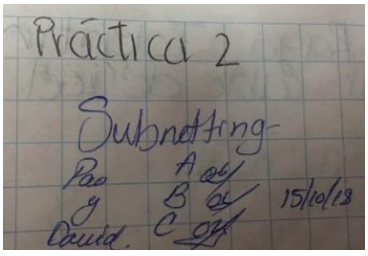
What we're trying to do with this report, apart from document and describe the code, is explaining why the code works and what's the theory behind it, in this case *Subnetting*. We'll also show the results and, more than analyze, try to give a deep understanding of what's been displayed.

By:

Paola Raya Tolentino and Jose David Portilla Martinez

Professor:
MSc. Nidia Asunción Cortez Duarte

October 2018



Index

Content

Introduction:

This document provides information about the making of subnetting practice, the ways that we can choose depending of the needs that we have, as the number of subnetworks you need, the number of hosts for subnetwork and the mask. You will learn how many subnetworks you can assign, how many subnetworks you can't assign and the way to get that information.

Literature review:

Subnetting

In short, subnetting is the practice of dividing a network (A, B or C Class) into more than two smaller networks, called subnets. It increases routing efficiency, enhances the security of the networks and reduces the sizes of the unused host that ultimately end up wasting space.

Basic Concepts

Before we dive in, it's necessary to have an idea of what an *IP Address*, a *Subnet* and a *Subnet Mask* are[1] there are some other necessary concepts, but we'll give them in later sections:

- *IP Address*: A logical numeric address that is assigned to every single computer, printer, switch, router or any other device that is part of a TCP/IP-based network
- *Subnet*: A separate and identifiable portion of an organization's network, typically arranged on one floor, building or geographical location
- *Subnet Mask*: A 32-bit number used to differentiate the network component of an IP address by dividing the IP address into a network address and host address

Binary Numbers

Without giving a thorough explanation of what binary numbers are or even mention any examples of their uses, here's a table that could be useful:

128	64	32	16	8	4	2	1
2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
10000000	01000000	00100000	00010000	00001000	00000100	00000010	00000001

Here's one example of how we used it:

> *Given an IP **205.112.45.60**, it could be written as:*
>
> ***11001101.01110000.00101101.00111100***

Here's another example:

> *Given a subnet mask **255.255.255.192**, it also could be written as **/26** and as:*
>
> ***11111111.11111111.11111111.11000000***

Understand Subnetting[2]

Each data link on a network must have a unique network ID, with every node on that link being a member of the same network. If you break a major network (Class A, B, or C) into smaller subnetworks, it allows you to create a network of interconnecting subnetworks. Each data link on this network would then have a unique network/subnetwork ID. Any device, or gateway, that connects *n* networks/subnetworks has *n* distinct IP addresses, one for each network / subnetwork that it interconnects.

To subnet a network, extend the natural mask with some of the bits from the host ID portion of the address to create a subnetwork ID. For example, given a Class C network of 204.17.5.0 which has a natural mask of 255.255.255.0, you can create subnets in this manner:

```
204.17.5.0 -      11001100.00010001.00000101.00000000
255.255.255.224 - 11111111.11111111.11111111.11100000
                  -------------------------|sub|----
```

By extending the mask to be 255.255.255.224, you have taken three bits (indicated by "sub" and these are the borrowed bits) from the original host portion of the address and used them to make subnets. With these three bits, it is possible to create eight subnets. With the remaining five host ID bits, each subnet can have up to 32 host addresses, 30 of which can be assigned to a device *since host ids of all zeros or all ones are not allowed* (it is very important to remember this). So, these subnets have been created:

```
204.17.5.0/27     host address range 1 to 30
204.17.5.32/27    host address range 33 to 62
204.17.5.64/27    host address range 65 to 94
204.17.5.96/27    host address range 97 to 126
204.17.5.128/27   host address range 129 to 158
204.17.5.160/27   host address range 161 to 190
204.17.5.192/27   host address range 193 to 222
204.17.5.224/27   host address range 225 to 254
```

Caveats or: Some Things You Need to Have in Mind

If you have an IP Address of any Class (*A, B* or *C*):

- You'll need at least 4 host for each network or subnet
- You'll need at least 1 subnet (in this case is the normal network)
- With the CIDR (Classless Inter-Domain Routing) notation, you'll get to have a */30* mask as a limit and at least a */8* mask
- In case you didn't know, taking the first byte:
  - ✓ *Class A*: 0 – 127
  - ✓ *Class B*: 127 – 191
  - ✓ *Class C*: 192 – 223

If you have a *Class A* IP Address*:*

- You'll have as much as 2^{24}-2 = 16777214 host per subnet
- You'll have as much as 2^{22} = 4194304 subnets
- Borrowed bits go from 1 to 22
- Host bits go from 2 to 24

If you have a *Class B* IP Address*:*

- You'll have as much as 2^{16}-2 = 65534 host per subnet
- You'll have as much as 2^{14} = 16384 subnets
- Borrowed bits go from 1 to 14
- Host bits go from 2 to 16

If you have a *Class C* IP Address*:*

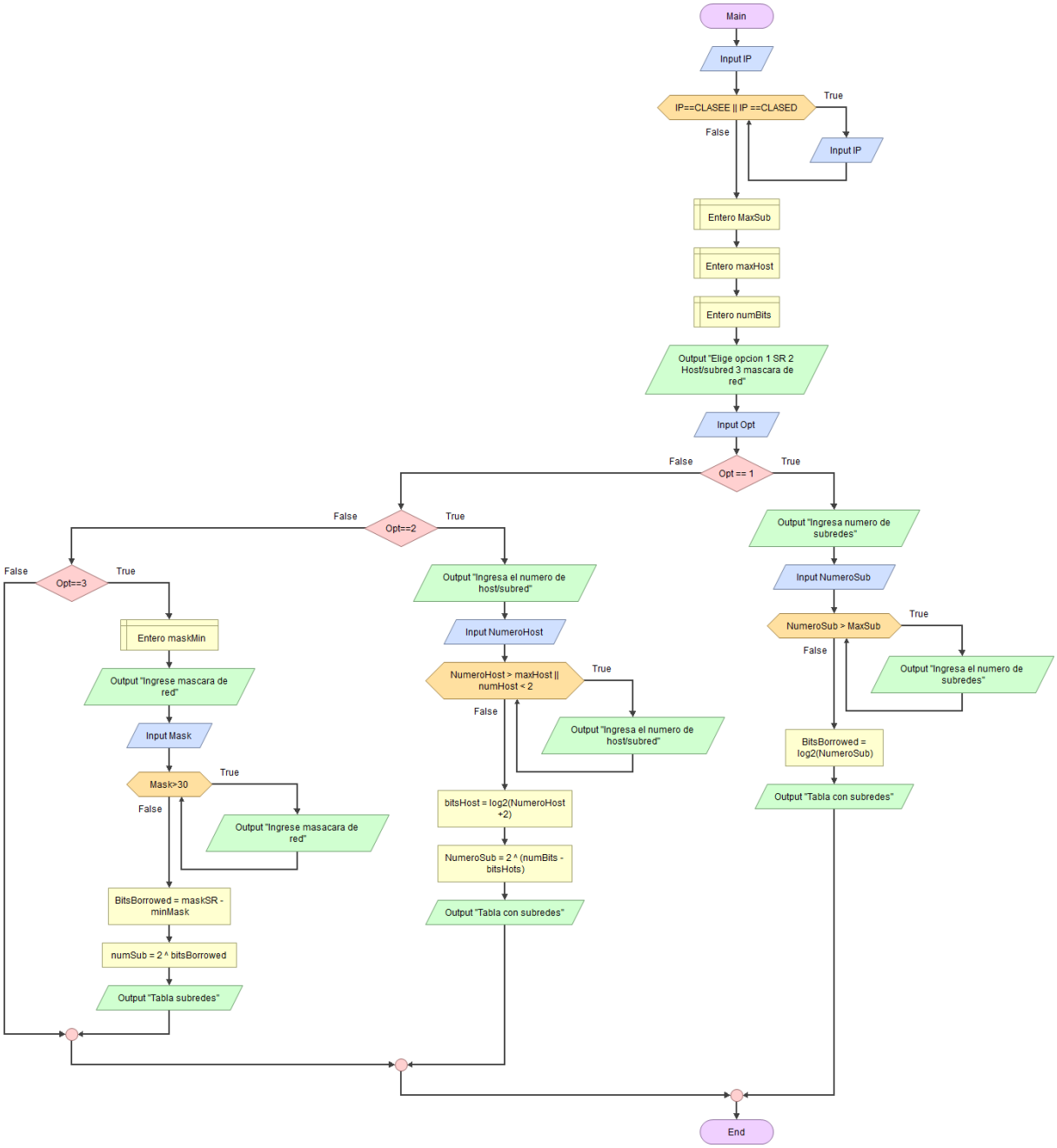
- You'll have as much as 2^{8}-2 = 254 host per subnet
- You'll have as much as 2^{6} = 64 subnets
- Borrowed bits go from 1 to 6
- Host bits go from 2 to 8

Note: You can't use both limits at the same time.

Software:

- Python 3.5.6
- Pretty Table for Python: https://pypi.org/project/PrettyTable/

Procedure:



The procedure that we use to do this practice was based in many validations, because we need to look in all the possibilities that can exist, and all the validations are the following:

- Cheek how many subnetworks the class can it have.
- How bits borrowed can use.
- How many hots/subnetworks can we use.
- Class D and E: if the IP class was one of this the program ask again for another IP address.

Results

To run: open CMD within the folder where the source code is and as it's shown below:

C:\Users\MyComputer> ***python subnetCalculator.py***

After that, the program will ask you for an IP address in the screen, then you can see every option that the program can show, as seen in *figure a.*

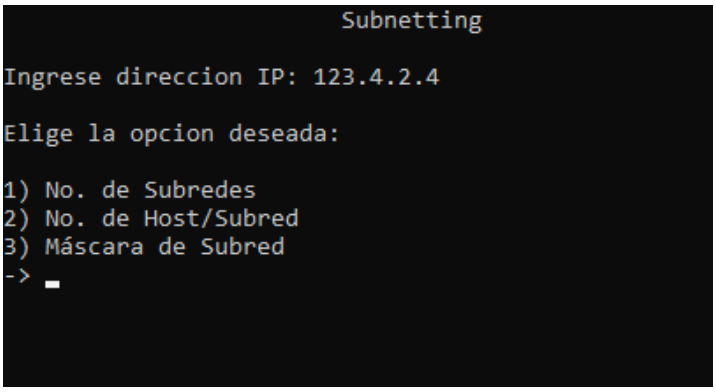


Figure a

For the case 1, we are going to use the IP address *123.4.2.4* and we're going to select the first option, that is "No. de Subredes", we ask the program to show us *16* subnets, then it'll show us all the information of the IP address that we put, as shown in *figure b*.

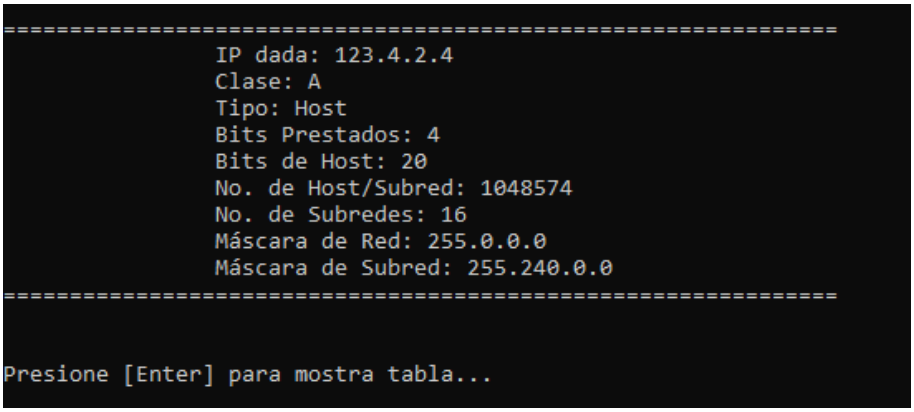
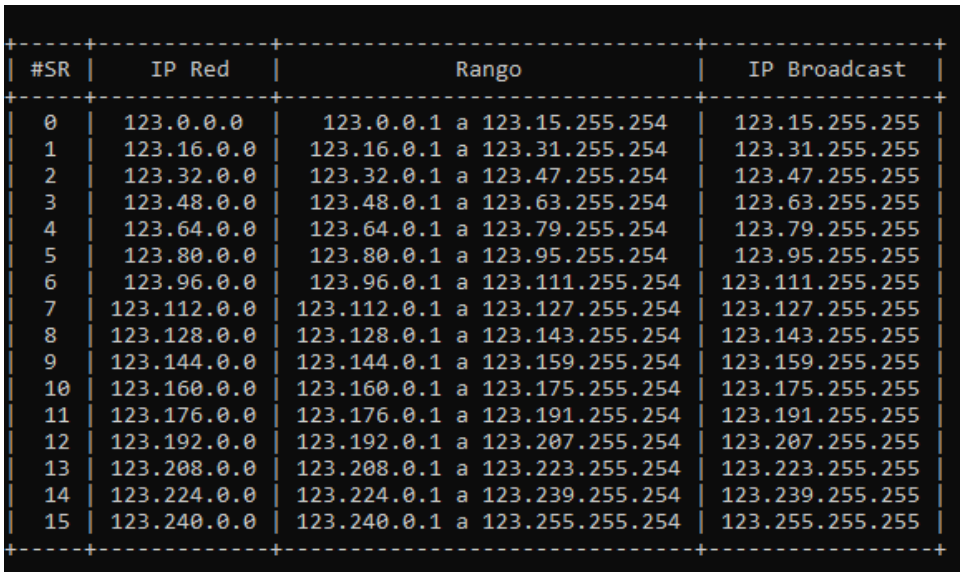


Figure b

Then we press enter and will see all 16 subnets, it can be seen in *figure c*



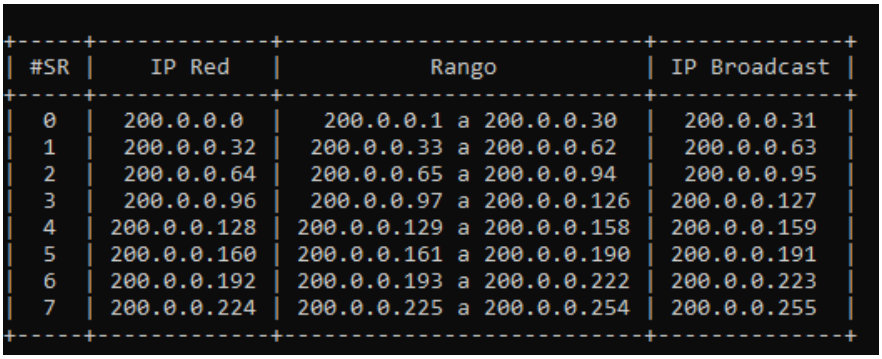
#SR	IP Red	Rango	IP Broadcast
0	123.0.0.0	123.0.0.1 a 123.15.255.254	123.15.255.255
1	123.16.0.0	123.16.0.1 a 123.31.255.254	123.31.255.255
2	123.32.0.0	123.32.0.1 a 123.47.255.254	123.47.255.255
3	123.48.0.0	123.48.0.1 a 123.63.255.254	123.63.255.255
4	123.64.0.0	123.64.0.1 a 123.79.255.254	123.79.255.255
5	123.80.0.0	123.80.0.1 a 123.95.255.254	123.95.255.255
6	123.96.0.0	123.96.0.1 a 123.111.255.254	123.111.255.255
7	123.112.0.0	123.112.0.1 a 123.127.255.254	123.127.255.255
8	123.128.0.0	123.128.0.1 a 123.143.255.254	123.143.255.255
9	123.144.0.0	123.144.0.1 a 123.159.255.254	123.159.255.255
10	123.160.0.0	123.160.0.1 a 123.175.255.254	123.175.255.255
11	123.176.0.0	123.176.0.1 a 123.191.255.254	123.191.255.255
12	123.192.0.0	123.192.0.1 a 123.207.255.254	123.207.255.255
13	123.208.0.0	123.208.0.1 a 123.223.255.254	123.223.255.255
14	123.224.0.0	123.224.0.1 a 123.239.255.254	123.239.255.255
15	123.240.0.0	123.240.0.1 a 123.255.255.254	123.255.255.255

Figure c

Now we are going to select the second option, “No. de host/subred”, in this case we use the IP address *200.0.0.3* and ask *23* hosts per subnet, like the previous option, the program is going to show us all the information of the IP address. We can see it in *figure d.*

Figure d

And the table for the subnets is shown in *figure e*.



#SR	IP Red	Rango	IP Broadcast
0	200.0.0.0	200.0.0.1 a 200.0.0.30	200.0.0.31
1	200.0.0.32	200.0.0.33 a 200.0.0.62	200.0.0.63
2	200.0.0.64	200.0.0.65 a 200.0.0.94	200.0.0.95
3	200.0.0.96	200.0.0.97 a 200.0.0.126	200.0.0.127
4	200.0.0.128	200.0.0.129 a 200.0.0.158	200.0.0.159
5	200.0.0.160	200.0.0.161 a 200.0.0.190	200.0.0.191
6	200.0.0.192	200.0.0.193 a 200.0.0.222	200.0.0.223
7	200.0.0.224	200.0.0.225 a 200.0.0.254	200.0.0.255

Figure e

Now for the last option we are going to use the IP address *190.0.0.0*, but now we need to enter de subnet mask in the CIDR notation, for example, if we want the subnet mask be *255.255.192.0*, we're going to type it like: */18*. Then program will give us, again, all information about the IP address and the table with the subnetworks. It's shown in *figure d.*

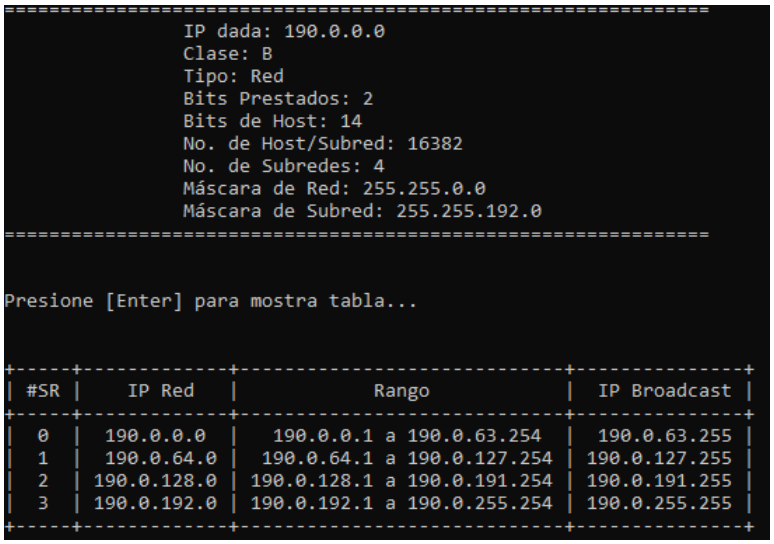


Figure d

Discussion:

How can we interpret the results we just saw? With the first part of the results, more precisely, the information of the IP Address, we can break down the next part, the table with all the subnets needed.

Using a different example than before, the IP Address *192.0.0.0* and, whether *3 or 4* subnets, *32* to *62* host or a */26* subnet mask:

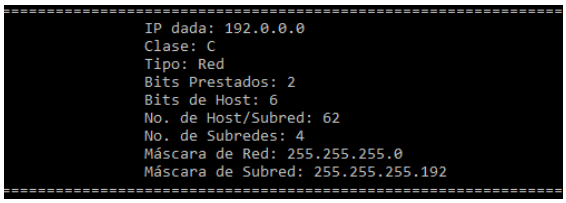


Figure e: IP Address Information

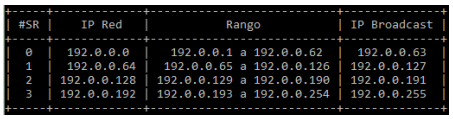


Figure f: Subnets Table

As said, using the table above (*figure e*) we can construct the table shown in the *figure f*, "How?" you may ask, well, let's break it in three cases (just like in our program):

With the Number of Subnets

In this case we need 4 subnets, first, is 4 a correct number? Well, it's on the table so it is correct! But how do we know it is? This number must be such that it can be display as 2^n (remember *n*, it's important), so $4 = 2^2$, this means it's correct. In the case that we could've been given a *3* and we'd must have used *4* again, because we need a number that covers up the number three in its entirely while been able to be display as 2^n.

So now what? How can we know how many hosts can be in each subnet? For this we'll need *n*, remember *n*? It represents the number of borrowed bits, each class of IP has its net bits and host bits (for A: 8 and 24, for B: 16 and 16, and for C: 24 and 8), but when subnetting, we take some borrowed bits, so we can divide the network into a determined number of subnets, that numbers it's represented by 2^n, the remaining bits stay as host bits, this are represented by *m*. So, in order to know how many hosts each subnet will contain we'll need to make this operation: 2^m-2. Why that minus 2 you ask? Each network, as well in each subnet, must own a network (in this case subnet) address and a broadcast address (we'll explain these in a moment) by default. In case you didn't notice, *m* is obtained by extracting the bits borrowed from the default host bits, so for this example, the number of hosts per subnet will be: $2^{8-2} - 2 = 62$.

Now the trickier part, everything else, the upside to this is that is the same in the other cases:

(Notation)
Network Bits
Borrowed Bits
Host Bits

Let's begin with the subnet address, this is first address in the network and it is used for identification of the network segment[3], one way to obtain it is turning off (set to 0) all of the host bits of the given IP address. We could identify each subnet with its borrowed bits because it helps us number them:

1st subnet address:	192.0.0.0	11000000.00000000.00000000.00000000
2nd subnet address:	192.0.0.64	11000000.00000000.00000000.01000000
3rd subnet address:	192.0.0.128	11000000.00000000.00000000.10000000
4th subnet address:	192.0.0.192	11000000.00000000.00000000.11000000

For the broadcast address, which is the last address in the network, and it is used for addressing all the nodes in the network at the same time[3], to obtained we turn on (set to 1) every host bit:

1st broadcast address:	192.0.0.63	11000000.00000000.00000000.00111111
2nd broadcast address:	192.0.0.127	11000000.00000000.00000000.01111111
3rd broadcast address:	192.0.0.191	11000000.00000000.00000000.10111111
4th broadcast address:	192.0.0.255	11000000.00000000.00000000.11111111

Now finally, to find the range of each subnet we only need their subnet and broadcast addresses, we add one to the subnet address and subtract one from the broadcast address:

1st subnet:	From 192.0.0.1 to 192.0.0.62
2nd subnet:	From 192.0.0.65 to 192.0.0.126
3rd subnet:	From 192.0.0.129 to 192.0.0.190
4th subnet:	From 192.0.0.193 to 192.0.0.254

With the Number of Hosts per Subnet

Now given how many hosts per subnet we need, as in the fist case, is essential to check if this number is correct, and it's almost the same, but instead of verifying this: 2^n we'll verify: 2^m-2. Another way to see it that we didn't mention before it's through obtaining *m*, and then making the operation:

Let's say *H* it's a correct number of hosts per subnet as it satisfies:

$$H = 2^m - 2$$

We need *m*, so we solve the equation for *m*:

$$m = log_2(H + 2)$$

As the given *H* it's probably not a correct number of hosts we need the least integer greater than or equal to m, thus applying the ceil function:

$$m = \lceil log_2(H + 2) \rceil$$

For this example:

$$m = \lceil log_2(62 + 2) \rceil = 6$$

Side note: for the previous case, if we apply this approach we'll end up with:

$$n = \lceil log_2(S) \rceil$$

With *S* the number of subnets needed.

Now with this equation we can both verify and obtain the number of hosts per subnet we need, but now we need in how many subnets the given address will divided into, it's a similar process than before, we'll subtract the host bits from the default host bits so that we can obtain *n* and then apply, $2^{8-6} = 4$. **The next steps are exactly like before.**

With the Subnet Mask

In one of the first sections we mention the CIDR notation for subnet and network masks, which is represent as: */SM or /NM*, being *SM or NM*, the number of bits turn on (set to 1) from left to right, so given the mask */26*:

11111111. 11111111. 11111111.11000000

How can we get the borrowed bits (*n*) and the host bits (*m*) with the subnet mask? Easily, *n* is going to be *SM – NM*, the network mask is given by default for each class of address, in this case it's C so the network mask it's */24*, therefore *n = 26 – 24 = 2*. With *m* is an easier process, we simple subtract *SM* from all the bits in the mask, thus *m = 32 – 26 = 6*.

Now we can obtain the number of subnets how many hosts they'll need, $4 = 2^2$ *and* $2^{8-2} - 2 = 62$ respectively. Finally, we do as the first time.

Conclusions:

There isn't just what thing that can be applied to a real-life situation, the use of subnetting is extremely important to manage all the devices that must be in a network, although this technique could be really inefficient, it also has a lot to offer, such as the use of subnets with an enormous amount of host. At the start of making this practice, we stumble across a big number of issues, especially when applying what we just did in class C to other classes, even if it was really useful at the beginning, we had to give it a different approach, not entirely, but different nevertheless.

From this came the biggest flaw in our code, the efficiency, when you get to code, it can be seen that in some functions we used a lot of operations with strings, instead of using bitwise operators, such operator can be such a reducer of needless operations, and we used as much as we can, but out code isn't replete with this.

References:

- **[1] Techopedia.com.** (2018). 8 Steps to Understanding IP Subnetting. [online] Available at: https://www.techopedia.com/6/28587/internet/8-steps-to-understanding-ip-subnetting [Accessed 30 Sep. 2018]
- **[2] Support, T., Routing, I. and TechNotes, T.** (2018). IP Addressing and Subnetting for New Users. [online] Cisco. Available at: https://www.cisco.com/c/en/us/support/docs/ip/routing-information-protocol-rip/13788-3.html [Accessed 30 Sep. 2018].
- **[3] Ladu.htk.tlu.ee.** (2018). Network and Broadcast Address. [online] Available at: http://ladu.htk.tlu.ee/erika/taavi/doc2/network_and_broadcast_address.html [Accessed 30 Sep. 2018].
- **Cope, S.** (2018). Subnetting and Subnet Masks Explained. [online] Steves-internet-guide.com. Available at: http://www.steves-internet-guide.com/subnetting-subnet-masks-explained/ [Accessed 30 Sep. 2018].

Code

Main code

```
from prettytable import PrettyTable
from netUtilities import *

if __name__ == '__main__':
    rep = 1
    while (rep):
        ipClass = -1
        while (ipClass == 1 or classIP(ipAddress[1]) == 1 or classIP(ipAddress[2]) == 1 or classIP(ipAddress[3]) == -1):
            system('cls')
            print ("\t\t\tSubnetting\n")
            print("Ingrese direccion IP: ", end = '')

            auxIP = input()
            ipAddress = [int(i) for i in ((auxIP).split('.'))]
            ipClass = classIP(ipAddress[0])

        if (ipClass != 'D' and ipClass != 'E'):
            print("\nElige la opcion deseada:\n\n1) No. de Subredes\n2) No. de Host/Subred\n3) Máscara de Subred")
            opt = -1
            while (opt < 1 or opt > 3):
                print ("-> ", end ='')
                opt = int(input())

            maxSub = asBigS(ipClass)
            maxHost = asBigH(ipClass)
            numBits = howBit(ipClass)
            staticByte = howByte(ipClass)

            table = PrettyTable(['#SR', 'IP Red', 'Rango', 'IP Broadcast']) #Se crea una tabla

            if (opt == 1):
                numSub = maxSub + 1
                while (numSub > maxSub): #Verifica que no ingrese un número más grande que el máximo
                    print("\nIngresa el No. de Subredes: ", end = '')
                    numSub = int(input())
                bitsBorrowed = nearPower(numSub)
                show(auxIP, ipClass, typeIP(ipAddress, ipClass), bitsBorrowed, numBits - bitsBorrowed, ((2 ** (numBits - bitsBorrowed)) - 2), (2 ** bitsBorrowed))
                for i in range (0, (2 ** bitsBorrowed)): #Por cada subred se iran agregando las filas con: #SB, IP Red, Rango e IP Broadcast
                    table.add_row([i, '.'.join([str(i) for i in ipAddress[:staticByte]]) + '.' + byteNet(i, bitsBorrowed, ipClass, 0), '.'.join([str(i) for i in ipAddress[:staticByte]]) + '.' + byteNet(i, bitsBorrowed, ipClass, 1) + ' a ' + '.'.join([str(i) for i in ipAddress[:staticByte]])+ '.' + byteBro(i, bitsBorrowed, ipClass, 1), '.'.join([str(i) for i in ipAddress[:staticByte]]) + '.' + byteBro(i, bitsBorrowed, ipClass, 0)])

            elif (opt == 2):
                numHost = -1
                while (numHost > maxHost or numHost < 2): #Verifica que no se ingrese ni más ni menos host por subred
                    print("\nIngresa el No. de Host/Subred: ", end = '')
                    numHost = int(input())
                bitsHots = nearPower(numHost + 2)
                numSub = 2 ** (numBits - bitsHots)
                show(auxIP, ipClass, typeIP(ipAddress, ipClass), numBits - bitsHots, bitsHots, ((2 ** bitsHots) - 2), numSub)
                for i in range (0, numSub): #Por cada subred se iran agregando las filas con: #SB, IP Red, Rango e IP Broadcast
                    table.add_row([i, '.'.join([str(i) for i in ipAddress[:staticByte]]) + '.' + byteNet(i, numBits - bitsHots, ipClass, 0), '.'.join([str(i) for i in ipAddress[:staticByte]]) + '.' + byteNet(i, numBits - bitsHots, ipClass, 1) + ' a ' + '.'.join([str(i) for i in ipAddress[:staticByte]]) + '.' + byteBro(i, numBits - bitsHots, ipClass, 1), '.'.join([str(i) for i in ipAddress[:staticByte]]) + '.' + byteBro(i, numBits - bitsHots, ipClass, 0)])

```

```
            elif (opt == 3):
                minMask = asShort(ipClass)
                maskSR = -1
                while (maskSR > 30 or maskSR < minMask): #Verifica que la máscara ingresada este dentro de los limites
                    print ("\nIngresa la Máscara de Subred: ", end = '')
                    aux = input()
                    maskSR = int(aux[1:])
                bitsBorrowed = maskSR - minMask
                numSub = 2 ** bitsBorrowed
                show(auxIP, ipClass, typeIP(ipAddress, ipClass), bitsBorrowed, numBits - bitsBorrowed, ((2 ** (numBits - bitsBorrowed)) - 2), (2 ** bitsBorrowed))
                for i in range (0, numSub): #Por cada subred se iran agregando las filas con: #SB, IP Red, Rango e IP Broadcast
                    table.add_row([i, '.'.join([str(i) for i in ipAddress[:staticByte]]) + '.' + byteNet(i, bitsBorrowed, ipClass, 0), '.'.join([str(i) for i in ipAddress[:staticByte]]) + '.' + byteNet(i, bitsBorrowed, ipClass, 1) + ' a ' + '.'.join([str(i) for i in ipAddress[:staticByte]]) + '.' + byteBro(i, bitsBorrowed, ipClass, 1), '.'.join([str(i) for i in ipAddress[:staticByte]]) + '.' + byteBro(i, bitsBorrowed, ipClass, 0)])

            print ("\nPresione [Enter] para mostra tabla...", end = '')
            input()
            print ("\n")
            print (table) #Muestra tabla

        else:
            system('cls')
            print ("\n================================================================")
            print ("\t\tIP dada: %s" % ('.'.join([str(i) for i in ipAddress])))
            print ("\t\tClase: %s" % (ipClass))
            print ("================================================================\n")

        print ("\nSi desea repetir teclee [1], si no [0]: ", end = '')
        rep = int(input())
```

Functions

```
from os import system
import math

def byteNet(i, bitsBorrowed, ipClass, rangeI): #Regresa los bytes de red y host
    if (ipClass == 'A'):
        if (bitsBorrowed <= 8):
            return str(i << (8 - bitsBorrowed)) + '.0.' + str (0 + rangeI)
        elif (bitsBorrowed <= 16):
            aux = bin(i)[2:]
            preSeparate = ''.join([str(i - i) for i in range(0, bitsBorrowed - len(aux))]) + aux
            fByte = preSeparate[:8]
            sByte = preSeparate[8:] + ''.join([str(i - i) for i in range(0, 8 - len(preSeparate[8:]))])
            return str(int(fByte, 2)) + '.' + str(int(sByte, 2)) + '.' + str (0 + rangeI)
        else:
            aux = bin(i)[2:]
            preSeparate = ''.join([str(i - i) for i in range(0, bitsBorrowed - len(aux))]) + aux
            fByte = preSeparate[:8]
            sByte = preSeparate[8:16]
            tByte = preSeparate[16:] + ''.join([str(i - i) for i in range (0, 8 - len(preSeparate[16:]))])
            return str(int(fByte, 2)) + '.' + str(int(sByte, 2)) + '.' + str (int(tByte, 2) + rangeI)

    elif (ipClass == 'B'):
        if (bitsBorrowed <= 8):
            return str(i << (8 - bitsBorrowed)) + '.' + str (0 + rangeI)
        else:
            aux = bin(i)[2:]
            preSeparate = ''.join([str(i - i) for i in range(0, bitsBorrowed - len(aux))]) + aux
            fByte = preSeparate[:8]
            sByte = preSeparate[8:] + ''.join([str(i - i) for i in range(0, 8 - len(preSeparate[8:]))])
            return str(int(fByte, 2)) + '.' + str(int(sByte, 2) + rangeI)

```

```
    elif (ipClass == 'C'):
        return str((i << (8 - bitsBorrowed)) + rangeI)

def byteBro(i, bitsBorrowed, ipClass, rangeS): #Regresa los bytes de broadcast y host
    if (ipClass == 'A'):
        if (bitsBorrowed <= 8):
            return str(((i << (8 - bitsBorrowed)) + ((2 ** (8 - bitsBorrowed)) - 1))) + '.255.' + str (255 - rangeS)
        elif (bitsBorrowed <= 16):
            aux = bin(i)[2:]
            preSeparate = ''.join([str(i - i) for i in range(0, bitsBorrowed - len(aux))]) + aux
            fByte = preSeparate[:8]
            sByte = preSeparate[8:] + ''.join([str(i//i) for i in range(1, 9 - len(preSeparate[8:]))])
            return str(int(fByte, 2)) + '.' + str(int(sByte, 2)) + '.' + str (255 - rangeS)
        else:
            aux = bin(i)[2:]
            preSeparate = ''.join([str(i - i) for i in range(0, bitsBorrowed - len(aux))]) + aux
            fByte = preSeparate[:8]
            sByte = preSeparate[8:16]
            tByte = preSeparate[16:] + ''.join([str(i//i) for i in range (1, 9 - len(preSeparate[16:]))])
            return str(int(fByte, 2)) + '.' + str(int(sByte, 2)) + '.' + str (int(tByte, 2) - rangeS)

    elif (ipClass == 'B'):
        if (bitsBorrowed <= 8):
            return str((i << (8 - bitsBorrowed)) + ((2 ** (8 - bitsBorrowed)) - 1)) + '.' + str (255 - rangeS)
        else:
            aux = bin(i)[2:]
            preSeparate = ''.join([str(i - i) for i in range(0, bitsBorrowed - len(aux))]) + aux
            fByte = preSeparate[:8]
            sByte = preSeparate[8:] + ''.join([str(i//i) for i in range(1, 9 - len(preSeparate[8:]))])
            return str(int(fByte, 2)) + '.' + str(int(sByte, 2) - rangeS)

    elif (ipClass == 'C'):
        return str(((i << (8 - bitsBorrowed)) + ((2 ** (8 - bitsBorrowed)) - 1)) - rangeS)

def howBit(ipClass): #Regresa que tantos bits de host por default tiene una IP
    return (24 if (ipClass == 'A') else (16 if (ipClass == 'B') else 8))

def howByte(ipClass): #Regresa que tantos bytes de red por default tiene una IP
    return (1 if (ipClass == 'A') else (2 if (ipClass == 'B') else 3))

def nearPower(numSH): #Dado un No de subredes o No de Host/Red deseados, regresa la expotente tal que 2^n = No cubre alguno de los dos.
    return math.ceil(math.log(numSH, 2))

def asBigS(ipClass): #Regresa el número máximo de subredes dada una IP
    return (((2 ** 22) - 2) if (ipClass == 'A') else (((2 ** 14) - 2) if (ipClass == 'B') else ((2 ** 6) - 2)))

def asBigH(ipClass): #Regresa el número máximo de host/subred dada una IP
    return (((2 ** 24) - 2) if (ipClass == 'A') else (((2 ** 16) - 2) if (ipClass == 'B') else ((2 ** 8) - 2)))

def asShort(ipClass): #Regresa un número tal que sea el minimo para mostrar una máscara de subred en el formato /N
    return ((8) if (ipClass == 'A') else ((8 * 2) if (ipClass == 'B') else (8 * 3)))

def maskDefault(ipClass): #Regresa la máscara de red que tiene por default cada ip dependiendo su clase
    return ("255.0.0.0" if (ipClass == 'A') else ("255.255.0.0" if (ipClass == 'B') else "255.255.255.0"))

def maskSubred(ipClass, bitsBorrowed): #Regresa la máscara de subred
    if(ipClass == 'A'):
        if (bitsBorrowed <=8):
            return "255." + str (((2 ** bitsBorrowed) - 1) << (8 - bitsBorrowed)) + '.0.0'
        elif (bitsBorrowed <= 16):
            return "255.255." + str (((2 ** (8 - (16 - bitsBorrowed))) - 1) << (8 - (8 - (16 - bitsBorrowed)))) + '.0'
        else:
            return "255.255.255." + str (((2 ** (8 - (24  - bitsBorrowed))) - 1) << (8 - (8 - (24 - bitsBorrowed))))

    elif(ipClass == 'B'):
        if(bitsBorrowed <= 8):
            return "255.255." + str (((2 ** bitsBorrowed) - 1) << (8 - bitsBorrowed)) + '.0'
        else:
```

```
            return "255.255.255." + str (((2 ** (8 - (16 - bitsBorrowed))) - 1) << (8 - (8 - (16 - bitsBorrowed))))

    elif(ipClass == 'C'):
        return "255.255.255." + str (((2 ** bitsBorrowed) - 1) << (8 - bitsBorrowed))

def typeIP(ipAddress, ipClass):
    if (ipClass == 'A'):
        fByte = bin(ipAddress[1])[2:]
        sByte = bin(ipAddress[2])[2:]
        tByte = bin(ipAddress[3])[2:]
        if (fByte == (len(sByte) * fByte[0]) and sByte == (len(sByte) * sByte[0]) and tByte == (len(tByte) * tByte[0]) and sByte[0] == tByte[0]):

            if (sByte[0] == '0'):
                return "Red"
            else:
                return "Broadcast"
        else:
            return "Host"

    elif (ipClass == 'B'):
        sByte = bin(ipAddress[2])[2:]
        tByte = bin(ipAddress[3])[2:]
        if (sByte == (len(sByte) * sByte[0]) and tByte == (len(tByte) * tByte[0]) and sByte[0] == tByte[0]):
            if (sByte[0] == '0'):
                return "Red"
            else:
                return "Broadcast"
        else:
            return "Host"

    elif (ipClass == 'C'):
        tByte = bin(ipAddress[3])[2:]
        if (tByte == (len(tByte) * tByte[0])):
            if (tByte[0] == '0'):
                return "Red"
            else:
                return "Broadcast"
        else:
            return "Host"

def classIP(IP): #Regresa la clase dada una IP
    if (IP & 256):
        return -1
    if (IP & 128):
        if (IP & 64):
            if (IP & 32):
                if (IP & 16):
                    if (IP & 8):
                        if (IP & 4):
                            if (IP & 2):
                                if (IP & 1):
                                    return "E"
                                else:
                                    return -1
                            else:
                                return "E"
                        else:
                            return "E"
                    else:
                        return "E"
                else:
                    return "D"
            else:
                return "C"
        else:
            return "B"
    else:
        return "A"
```

```
167.
168. def show(ipAddress, ipClass, ipType, bitsBorrowed, bitsHost, numHost, numSubn): #Muestra todo los argumentos dados
169.     system('cls')
170.     print ("\n================================================================")
171.     print ("\t\tIP dada: %s" % (ipAddress))
172.     print ("\t\tClase: %s" % (ipClass))
173.     print ("\t\tTipo: %s" % (ipType))
174.     print ("\t\tBits Prestados: %s" % (bitsBorrowed))
175.     print ("\t\tBits de Host: %s" % (bitsHost))
176.     print ("\t\tNo. de Host/Subred: %d" % (numHost))
177.     print ("\t\tNo. de Subredes: %d" % (numSubn))
178.     print ("\t\tMáscara de Red: %s" % (maskDefault(ipClass)))
179.     print ("\t\tMáscara de Subred: %s" % (maskSubred(ipClass, bitsBorrowed)))
180.     print ("================================================================\n")
```