



**INSTITUTO POLITÉCNICO NACIONAL
ESCUELA SUPERIOR DE CÓMPUTO
REDES DE COMPUTADORAS**



“CACULADORA IP”

GRUPO: 2CV15

EL SIUUU TEAM

INTEGRANTES:

Fischer Salazar César Eduardo

López García José Eduardo

Meza Vargas Brandon David

PROFESOR: Fabián Gaspar Medina

índice

Introducción.....	3
Descripción del código comentado	4
Ejecución del programa.....	16
Conclusiones.....	23

Introducción

Dentro de las redes de computadoras, existen diferentes elementos que componen todo un esquema lógico que permite la comunicación entre dispositivos situados en áreas cercanas o lejanas; uno de dichos recursos es el direccionamiento IP, el cual funciona como un identificador unívoco dentro de la topología, ya sea para un equipo o host. Usualmente, se utiliza el direccionamiento IPv4 (dirección lógica de 4 bytes separados por un punto) para mantener la comunicación dentro de un sector, como una oficina, una escuela o una ciudad, y desde mucho tiempo se han empleado métodos para poder calcular y obtener tanto las direcciones IP como sus máscaras que requiere un sistema de red para cumplir con su propósito.

Los métodos de subneteo tradicional y VLSM se han encargado por mucho tiempo la tarea de generar y clasificar un sinfín de direcciones IP y máscaras que hoy en día nos otorgan ese identificador y además de observar la pertenencia una subred de otra; y con el paso de los años, se ha buscado la manera de que estos cálculos realizados a mano puedan tener una elaboración mucho más óptima, eficaz y que ayude a ocupar la menor cantidad de tiempo empleado para la planeación del sistema a realizar.

Tal motivo ha hecho que muchos programadores realicen calculadoras IP (online y ejecutables), precisamente para hacer más rápida esta tarea; en nuestro caso, la realización de esta práctica se ha llevado de la siguiente forma: la utilización del lenguaje JavaScript y el apoyo de vue para implementar todas las operaciones y funciones de la calculadora, así como el uso de este trabajo en la red con html y css para su interfaz gráfica; la cual va a generar todo el proceso de detectar la IP que el usuario ingrese, el prefijo, la cantidad de subredes y host que requerimos para realizar dichos cálculos, y de tal manera que nos devuelvan la clase de la dirección, las IP's, las submáscaras, 1ra IP útil, última IP útil y broadcast para que el trabajo del usuario esté listo.

Descripción del código comentado

A continuación se presenta el desarrollo del programa, en las mismas capturas se presenta toda la explicación, pues el código esta comentada línea por línea, de esta forma no será necesario incluir explicaciones de más.

En primer lugar, tenemos las funciones comunes de todas las partes de nuestro programa:

```
const ip = "192.0.0.0" //Variable que almacena la ip que ingresa usuario

let expa //Variable que guardara el exponente al que sera elevado el 2 para obtener subredes y hosts

let octetToModify = 3 //Variable para saber el octeto de la ip a modificar, por defecto se modifica el último

/**
 * Función que obtiene los octetos de la ip ingresada
 * @param ip la ip escrita por el usuario
 * @returns los octetos de la dirección ip ingresada
 */
const getOctets = (ip) => ip.split('.') map(octet => Number.parseInt(octet,10))

const ipOctets = getOctets(ip) // Ahora vamos a llamar a la función que calcula los octetos

/**
 * Función utilizada para validar una dirección IP ingresada por el usuario
 * hacemos uso de una expresión regular que indica si una IP es válida o no
 * @param ip la ip escrita por el usuario
 * @returns true si es ip válida o false en caso de no ser una ip válida
 */
const validateIP = (ip) => {
  // Testeamos la ip del usuario con la expresión regular
  if(/^(25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)\.^(25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)\.^(25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)$/i.test(ip))
    return true // True si es válida
  else return false // False si no es válida
}

/** You, 3 days ago • v2 and documentation
 * Función que arroja un mensaje si la ip no es valida
 * @param * No recibe parametros
 * @returns Mensaje indicando que no es una ip válida
 */
const isValid = () =>{
  //Si la ip no es válida arrojamos mensaje indicando que ingrese otra IP
  if(!validateIP(ip))
    console.log("Inserte una IP válida o que pertenezca a clase A, B o C");
}

isValid() // Llamada a la función que valida la IP
```

```

/**
 * Función que inicializa la ip ingresada por el usuario con su clase, mascara, bits para host, la primera subnet, etc.
 * @param ipOctects Los octetos de la ip ingresada por el usuario
 * @returns La ip con su información por defecto
 */
const getIpObject = (ipOctects) => {
  let ObjectIp = new Object() // Nuevo objeto para la IP

  if(ipOctects[0] >= 0 && ipOctects[0] <= 127){ // Si la ip es clase A se inicializan los valores
    ObjectIp = {
      netClass : "Clase A", // Asignamos la clase
      defaultMask : "255.0.0.0", // Asignamos la mascara por defecto
      maskBits : "11111111000000000000000000000000", // Asignamos los bits de la mascara
      hostBits : 24, // Los bits para host
      netBits : 8, // Los bits para subredes
      firstSubNet : [ipOctects[0], 0, 0, 0] // La primera subred de la direccion
    }
  }
  if(ipOctects[0] >= 128 && ipOctects[0] <= 191){ // Si la ip es clase B se inicializan los valores
    ObjectIp = {
      netClass : "Clase B", // Asignamos la clase
      defaultMask : "255.255.0.0", // Asignamos la mascara por defecto
      maskBits : "11111111111111111100000000000000", // Asignamos los bits de la mascara
      hostBits : 16, // Los bits para host
      netBits : 16, // Los bits para subredes
      firstSubNet : [ipOctects[0], ipOctects[1], 0, 0] // La primera subred de la direccion
    }
  }
  if(ipOctects[0] >= 192 && ipOctects[0] <= 223){ // Si la ip es clase C se inicializan los valores
    ObjectIp = {
      netClass : "Clase C", // Asignamos la clase
      defaultMask : "255.255.255.0", // Asignamos la mascara por defecto
      maskBits : "1111111111111111111111111100000000", // Asignamos los bits de la mascara
      hostBits : 8, // Los bits para host
      netBits : 24, // Los bits para subredes
      firstSubNet : [ipOctects[0], ipOctects[1], ipOctects[2], 0] // La primera subred de la direccion
    }
  }

  return ObjectIp // Retornamos la ip con sus atributos
}

const completeIP = getIpObject(ipOctects) // Mandamos a llamar la función que obtiene los detalles de la IP

/**
 * Función que calcula una constante que nos servirá para el calculo de las nuevas submascaras de las redes
 * @returns Retorna la cantidad total de los 4 bytes que tiene la red
 */
const auxSubmaskCalc = () =>{
  let res = 0; // Inicializamos en 0
  for(i=0; i<32; i++){ // Ciclo hasta el número de bits de la red
    res = res + Math.pow(2,i) // Vamos sumando de acuerdo a la potencia de i
  }
  return res // Retornamos el resultado
}

const res = auxSubmaskCalc() //Mandamos a llamar la función que nos retorna la constante auxiliar para calculo de submascaras nuevas

```

```

/**
 * Función que calcula el prefijo de la nueva submascara
 * @param powSubnet Es la potencia a la que está elevada la subred
 * @param netBits Los bits de red de la dirección
 * @returns El prefijo de la nueva submascara de la red
 */
const prefixFunc = ({ netBits }, powSubnet) => netBits + powSubnet //Se suman los bits de red con la potencia de la subred

/**
 * Función que calcula la nueva mascara
 * @param expo El exponente al que está elevado
 * @param res Resultado obtenido de la función auxSubmaskCalc
 * @returns la nueva mascara de la red
 */
const getNewMask = (expo, res) => {
  /*
   La nueva mascara esta obtenida en una sola línea, se desglosará para mejor entendimiento
   (res << expo) >>> 0 Primero se realiza un corrimiento de bits a la izquierda de acuerdo al exponente
   Posteriormente un corrimiento sin signo a la derecha para obtener la cadena de bits completa
   de la mascara
   .toString(2) Pasamos a binario la cantidad obtenida con las operaciones anteriores
   .match() Tomamos 8 bits de la cadena de 32 bits
   .map() Recorremos de 8 en 8 los bits de la mascara para convertirlos en un entero decimal
  */
  return ( ( ( res << expo ) >>> 0 ).toString(2)).match(/.{1,8}/g).map(byte => parseInt(byte, 2))
}

/**
 * Función que calcula el salto para ir calculando las subredes de la dirección requerido de acuerdo a los hosts, subredes o prefijo
 * @param {*} prefix El prefijo de la red
 * @param {*} newMask La nueva mascara calculada anteriormente
 * @param {*} netClass La clase de la dirección Ip
 * @returns Retorna el salto para cada subred
 */
const hopFunc = ({ netClass }, prefix, newMask) => {
  let hop
  if(netClass == "Clase A") // Si la clase es A:
    if (prefix >= 8 && prefix <= 16){ // Si el prefijo está entre 8 y 16 el salto será de 256 menos el segundo octeto
      hop = 256 - newMask[1] // El octeto a modificar será el segundo
      octectToModify = 1
    } else if(prefix >= 17 && prefix <= 24){ // Si el prefijo está entre 17 y 24:
      hop = 256 - newMask[2] // El salto será 256 - el octeto 3 de la red
      octectToModify = 2 // El octeto a modificar será el 3
    } else hop = 256 - newMask[3] // En otro caso el salto será 256 menos el valor del último octeto

  if(netClass == "Clase B") // Si la clase es B:
    if (prefix >= 16 && prefix <= 24){ // Si el octeto está entre 16 y 24
      hop = 256 - newMask[2] // El salto será 256 - el octeto 3 de la red
      octectToModify = 2 // El octeto a modificar será el 3
    } else hop = 256 - newMask[3] // En otro caso el salto será 256 menos el valor del último octeto

  if(netClass == "Clase C") // Si la clase es C:
    hop = 256 - newMask[3] // En otro caso el salto será 256 menos el valor del último octeto

  return hop //Retornamos el salto resultante
}

```



```

/**
 * Función que calcula las subredes de la red
 * @param hop El salto de la red
 * @param totalSubnetworks La cantidad total de subredes que tiene la red
 * @param firstSubnet La primera subnet de la red
 * @param netClass La clase de la red
 * @returns La cantidad total de subredes que tiene la red expresada con sus octetos
 */
const subnetsCalc = ({ firstSubNet, netClass }, hop, totalSubnetworks) => {
  // En un arreglo temporal vamos a guardar arreglos de subredes que serán modificadas después
  const temporalNet = new Array(totalSubnetworks).fill(firstSubNet)
  let acc = 0 // Variable auxiliar de acumulador
  let acc2 = 0 // Variable auxiliar de acumulador
  let acc3 = 0 // Variable auxiliar de acumulador
  let subnets = [] // Arreglo que guardará las subredes

  // Por cada subred temporal se calculará la subred real
  subnets = temporalNet.map((subnet, index) => {
    if(index == 0) // Si pasamos por la primera subred la guardamos en el arreglo
      return subnet // Retornamos la primera subred

    const copy = ipOoctets.slice(0) // Realizamos una copia de los octetos de la red

    subnet = copy // Se copian los octetos en cada subred
    acc += hop // Se va acumulando el salto en el primer acumulador
    subnet[octectToModify] = acc // De acuerdo al octeto a modificar

    if(subnet[octectToModify] >= 256){ // Si el octeto a modificar sobrepasa los 255 se procede a modificar el anterior
      acc = 0 // Empezamos el acumular en 0 de nuevo
      acc2 += 1 // Para calcular las subredes vamos sumando uno al acumulador dos
      subnet[octectToModify] = 0 // El octeto a modificar se pondrá en 0
      subnet[octectToModify-1] = acc2 // Ahora si comenzamos con el octeto anterior a modificarlo
    }
    if(subnet[octectToModify-1] >= 256){ // Si el octeto a modificar-1 sobrepasa los 255 se procede a modificar el octeto-2
      acc = 0 // Reiniciamos el primer acumulador
      acc2 = 0 // Reiniciamos el segundo acumulador
      acc3 += 1 // Procedemos a aumentar el acumulador 3
      subnet[octectToModify-1] = 0 // Reiniciamos el octeto a modificar-1
      subnet[octectToModify-2] = acc3 // Comenzamos a modificar el octeto a modificar - 2
    }
    subnet[octectToModify] = acc // Finalmente el octeto a modificar será el acumulador
    subnet[octectToModify-1] = acc2 // El octeto-1 será el acumulador 2
    subnet[octectToModify-2] = acc3 // El octeto-2 será el acumulador 3

    if(netClass == "Clase A"){ // Si la clase es tipo A
      subnet = subnet.splice(0) // Copiamos la subnet
      subnet[0] = ipOoctets[0] // El primero octeto de la red será el primer octeto de la red ingresada
    }
    if(netClass == "Clase B"){ // Si la clase es tipo B
      subnet[0] = ipOoctets[0] // El primer octeto de cada subred será el primero de la red ingresada
      subnet[1] = ipOoctets[1] // El segundo octeto de cada subred será el segundo de la red ingresada
    }
    if(netClass == "Clase C"){ // Si la clase es tipo C
      subnet[0] = ipOoctets[0] // El primer octeto de cada subred será el primero de la red ingresada
      subnet[1] = ipOoctets[1] // El segundo octeto de cada subred será el segundo de la red ingresada
      subnet[2] = ipOoctets[2] // El tercer octeto de cada subred será el tercer de la red ingresada
    }

    return subnet // Retornamos cada subred
  })

  return subnets // Retornamos todas las subredes
}

```

```

/**
 * Función que calcula las direcciones de hosts para una subred específica
 * @param netClass La clase de la dirección de la IP
 * @param sub La subred elegida
 * @param gotHosts El número de hosts total
 * @returns Retorna las direcciones de host para la subred específica
 */
const hostCalc = ( { netClass }, sub, gotHosts) => {
  let acc = 0 // Variable auxiliar de acumulador
  let acc2 = 0 // Variable auxiliar de acumulador
  let acc3 = 0 // Variable auxiliar de acumulador

  // Variable temporal que guardara las direcciones de host sin calcular
  let temp = new Array(gotHosts).fill(sub)

  let hostXsubnet = [] // Arreglo que almacena las redes de host
  acc = sub[3] // Inicializamos el ultimo octeto
  acc2 = sub[2] // Inicializamos el tercer octeto
  acc3 = sub[1] // Inicializamos el segundo octeto

  // Recorremos cada red del arreglo temporal para ahora si calcular las redes de host
  hostXsubnet = temp.map((s) => {

    const copy = sub.slice(0) // Copiamos la subred
    s = copy // Se almacena la copia en cada subred

    acc += 1 // Vamos incrementando en uno el primer acumulador
    s[3] = acc // El ultimo octeto sera igual al primer acumulador

    /*
     Si el ultimo octeto rebasa los 255 se comienza a incrementar el acumulador 2 el cual sera
     lo que incremente el tercer octeto de la red de host. Este acumulador crecerá de uno en uno
    */
    if(s[3] >= 256){
      acc = 0
      acc2 += 1
      s[3] = 0
      s[2] = acc2
    }
  })
}

```



```

/*
    Si el tercer octeto rebasa los 255 se comienza a incrementar el acumulador 3 el cual sera
    lo que incremente el segundo octeto de la red de host. Este acumulador crecerá de uno en uno
*/
if( s[2] >= 256){
    acc = 0
    acc2 = 0
    acc3
    acc3 += 1
    s[2] = 0
    s[1] = acc3
}
s[3] = acc // El acumulador 1 será el ultimo octeto
s[2] = acc2 // El acumulador 2 será el tercer octeto
s[1] = acc3 // El acumulador 3 sera el segundo octeto

// Si la clase es tipo A no se cambia el primer octeto
if(netClass == "Clase A"){
    s = s.splice(0)
    s[0] = sub[0]
}
// Si la clase es tipo B no se cambian los primeros dos octetos
if(netClass == "Clase B"){
    s[0] = sub[0]
    s[1] = sub[1]
}
// Si la clase es tipo C no se cambian los primeros tres octetos
if(netClass == "Clase C"){
    s[0] = sub[0]
    s[1] = sub[1]
    s[2] = sub[2]
}

return s // Retornamos cada red de host
})
return hostXsubnet // Retornamos todas las redes de host
}
/**
 * Función que calcula la dirección de broadcast
 * @param hosts Lista que contiene las direcciones de host
 * @returns Retorna la dirección de broadcast para la subred
 */
const broadcastCalc = (hosts) =>{
    const broad = hosts[hosts.length-1] // Almacenamos la última dirección de host
    broad[3] = broad[3]+1 // Sumamos 1 al último octeto de la red, esta será la dir. broadcast
    return broad // Retornamos la dirección broadcast
}

const broadcastAdrrSubnet = broadcastCalc(hostsList) // Llamamos a la función que calcula la broadcast

```

Ahora bien, tenemos la parte donde se hacen los cálculos a partir de que el usuario ingresa el número de subredes que desea obtener.

```
//Usando subredes

//Restricciones

/**
 * Función que verifica que las sub redes requeridas por el usuario sean acordes al tipo de clase de la red
 * @param netClass La clase de la ip ingresada
 * @param requireSubnets El numero de subredes requeridas por el usuario
 */
const subnetsRestric = ( { netClass }, requireSubnets ) => {
  // Si es clase A y se requieren mas subredes de las que se pueden crear manda error
  if(netClass == "Clase A" && requireSubnets > 4194304)
    console.log("Inserte un número valido de subredes, inserte un valor de 1 a 4194304");

  // Si es clase B y se requieren mas subredes de las que se pueden crear manda error
  if(netClass == "Clase B" && requireSubnets > 16384)
    console.log("Inserte un número valido de subredes, inserte un valor de 1 a 16384");

  // Si es clase C y se requieren mas subredes de las que se pueden crear manda error
  if(netClass == "Clase C" && requireSubnets > 64)
    console.log("Inserte un número valido de subredes, inserte un valor de 1 a 64");
}

// Cantidad de subredes requeridas
const requireSubnets = 2

// Mandamos a llamar a la función de las restricciones
subnetsRestric(completeIP, requireSubnets)

const expoSubnets = Math.ceil(Math.log2(requireSubnets)) // Calculamos el exponente de la subred
const gotSubnets = Math.pow(2,expoSubnets) // Calculamos la cantidad de subredes totales
const hostPow = completeIP.hostBits - expoSubnets // Calculamos la potencia necesaria para calcular los hosts
const totalHostSubnet = Math.pow(2,hostPow)-2 // Calculamos el numero total de hosts que tendra cada subred
  //obteniendo nueva mascara
const newMaskSubnet = getNewMask(hostPow, res) // Mandamos a llamar a la funcion que calcula nueva mascara

const prefixSubnet = prefixFunc(completeIP ,expoSubnets) // Calculamos el prefijo

const hopSubnet = hopFunc(completeIP ,prefixSubnet, newMaskSubnet) // Calculamos el salto de cada subred
// Terminan calculos necesarios

const subnetListSubnet = subnetsCalc(completeIP ,hopSubnet, gotSubnets) // Calculamos la lista de sub redes
...subnetHost = subnetListSubnet[0]

console.log(subnetListSubnet);
console.log(subnetListSubnet[subnetListSubnet.length-1]);

const hostsList = hostCalc(completeIP ,...subnetHost, totalHostSubnet) // Calculamos la lista de hosts para una subred dada

console.log(hostsList[hostsList.length-1]);
console.log(hostsList);
```

En las siguientes capturas se presenta la parte de cuando el usuario elige hacer los cálculos a partir de los hosts:

```
let requireHosts = 1

//Restricciones
const hostRestric = ({ netClass }, requireHosts) => {

  // Si la clase es A y la cantidad de hosts es mayor a la posible regresamos error
  if( netClass == "Clase A" && requireHosts > 16777214 )
    console.log("Inserte una cantidad de Hosts válida, inserte un valor de 1 a 16777214");

  // Si la clase es B y la cantidad de hosts es mayor a la posible regresamos error
  if( netClass == "Clase B" && requireHosts > 65534 )
    console.log("Inserte una cantidad de Hosts válida, inserte un valor de 1 a 65534");

  // Si la clase es C y la cantidad de hosts es mayor a la posible regresamos error
  if( netClass == "Clase C" && requireHosts > 254 )
    console.log("Inserte una cantidad de Hosts válida, inserte un valor de 1 a 254");
}

hostRestric(completeIP, requireHosts) // Llamamos a la función que verifica los hosts

const expoHost = Math.ceil(Math.log2(requireHosts+2)) // Calculamos el exponente necesario para obtener los hosts

const gotHosts = Math.pow(2,expoHost)-2 // Calculamos la cantidad de hosts

let subnetPow = completeIP.hostBits - expoHost // Calculamos la potencia de la subred

const totalSubnetworksHost = Math.pow(2,subnetPow) // Calculamos el total de subredes

const newMaskHost = getNewMask(expoHost, res) // Calculamos la nueva máscara
const prefixHost = prefixFunc(completeIP, subnetPow) // Calculamos el prefijo

const hopHost = hopFunc(completeIP, prefixHost, newMaskHost)

const subnetList = subnetsCalc(completeIP, hopHost, totalSubnetworksHost) // Calculamos la lista de subredes
const subnetHost = subnetList[0]

console.log(subnetList);
console.log(subnetList[subnetList.length-1]);

const totalHost = hostCalc(completeIP, subnetHost, gotHosts) // Calculamos los hosts de una subred dada

console.log(totalHost[totalHost.length-1]);
console.log(totalHost);

const broadcastAddr = broadcastCalc(totalHost) // Calculamos la red de broadcast
```

Y finalmente cuando el usuario quiere hacer los cálculos a partir de ingresar el prefijo:

```
//Usando el prefijo

const requirePrefix = 24 // Prefijo requerido
//Restricciones

/**
 * Función que indica las restricciones para el prefijo ingresado por el usuario
 * @param netClass La clase de la red
 * @param requirePrefix El prefijo deseado por el usuario
 */
const prefixRestric = ({ netClass }, requirePrefix) => {
  // Si el prefijo no es valido se produce un mensaje de error
  if( requirePrefix < 8 || requirePrefix > 30 )
    console.log("Prefijo no valido, prueba con un valor de 8 a 30");

  // Si el prefijo no es valido para la clase B se produce un mensaje de error
  if( netClass == "Clase B" && requirePrefix < 16 )
    console.log("Prefijo no válido para clase B, inserte un valor de 16 a 30");

  // Si el prefijo no es valido para la clase C se produce un mensaje de error
  if( netClass == "Clase C" && requirePrefix < 24 )
    console.log("Prefijo no válido para clase C, inserte un valor de 24 a 30");
}

prefixRestric(completeIP, requirePrefix) // Llamamos a la funcion paara verificar el prefijo

const subnetPowPrefix = requirePrefix - completeIP.netBits // Calculamos la potencia para subredes
const hostPowPrefix = completeIP.hostBits - subnetPowPrefix // Calculamos la potencia para hosts

const newMaskPrefix = getNewMask(hostPowPrefix, res) // Calculamos la nueva mascara

const totalHostPrefix = Math.pow(2,hostPowPrefix)-2 // Calculamos el total de host

const totalSubnetsPrefix = Math.pow(2,subnetPowPrefix) // Total de subredes

const hopPrefix = hopFunc(completeIP, requirePrefix, newMaskPrefix) // Calculamos el salto

const subnetsListPrefix = subnetsCalc(completeIP, hopPrefix, totalSubnetsPrefix) // Calculamos las direcciones de subredes
console.log(subnetsListPrefix);
console.log(subnetsListPrefix[subnetsListPrefix.length-1]);

const hostPrefix = subnetsListPrefix[0]
const hostListPrefix = hostCalc(completeIP, hostPrefix, totalHostPrefix) // Calculamos las direcciones de hosts de una subred
console.log(hostListPrefix);
console.log(hostListPrefix[hostListPrefix.length-1]);

const broadcastAdrrPrefix = broadcastCalc(hostListPrefix) // Calculamos la dirección de broadcast
```

Como se menciona en la introducción, quisimos ir un paso más allá incluyendo una interfaz gráfica interactiva para el usuario y que nuestro proyecto sea desplegado en web haciendo uso de uno de los frameworks modernos más populares como lo es Vue.

A continuación se presentan las capturas de los elementos necesarios para las interfaces, al tener la estructura similar, el código es parecido.

Para cálculos a partir de las subredes:

```
<template>
  <div class="container-sub">
    <form class="form-container">
      <div class="text-field">
        <label>Dirección IP</label>
        <input v-model.lazy="ip" placeholder="Dirección IP" required @blur="isipvalid()">
      </div>
      <div class="text-field">
        <label>No. Subredes</label>
        <input v-model.lazy="requireNumber" type="number" placeholder="No. SUBREDES" required @blur="IpCalcs()">
      </div>
    </form>
    <div class="cards" v-if="cards.length > 1">
      <card
        v-for="(card, index) in cards" :key="index"
        :title="card.title"
        :icon="card.icon"
        :value="card.value.toString()"
      >
      </card>
      <button class="calc" @click="subnetsCalc()">calcular</button>
    </div>
    <section class="subnets-list" v-if="hasSubnets">
      <VTable>
        <title>Lista de Subredes</title>
        <list>subnetlistSubnet</list>
        <@modalShow="showModal = true">
        <calc>hostCalc</calc>
        <pre>prefixSubnet</pre>
      </VTable>
    </section>
    <modal v-if="showModal" @close="showModal = false" hosts="hostslist"
      :broad="broadcastAddrSubnet">
      <h3>custom header</h3>
    </modal>
  </div>
</template>
```


Para los cálculos a partir de los hosts:

```
<template>
  <div class="container-sub">
    <form class="form-container">
      <div class="text-field">
        <label>Dirección IP</label>
        <input v-model.lazy="ip" placeholder="Dirección IP" required @blur="isIpValid()">
      </div>
      <div class="text-field">
        <label>No. Hosts</label>
        <input v-model.lazy="requireNumber" type="number" placeholder="No. Hosts" required @blur="ipCalcs()">
      </div>
    </form>
    <div class="cards" v-if="cards.length > 1">
      <card
        v-for="(card, index) in cards" :key="index"
        :title="card.title"
        :icon="card.icon"
        :value="card.value.toString()"
      >
    </div>
    <button class="calc" @click="subnetsCalc()">calcular</button>
  </div>
  <section class="subnets-list" v-if="hasSubnets">
    <VTable
      title="Lista de Subredes"
      :list="subnetListSubnet"
      @modalShow="showModal = true"
      :calc="hostCalc"
      :pre="prefixSubnet"
    >
    </VTable>
  </section>
  <modal v-if="showModal" @close="showModal = false" :hosts="hostsList"
    :broad="broadcastAdrrSubnet"
  >
    <div>
      <h3>custom header</h3>
    </div>
  </modal>
</div>
</template>
```

Para los cálculos a partir del prefijo:

```
<template>
  <div class="container-sub">
    <form class="form-container">
      <div class="text-field">
        <label>Dirección IP</label>
        <input v-model.lazy="ip" placeholder="Dirección IP" required @blur="isIpValid()">
      </div>
      <div class="text-field">
        <label>Prefijo</label>
        <input v-model.lazy="requireNumber" type="number" placeholder="Prefijo" required @blur="ipCalcs()" >
      </div>
    </form>
    <div class="cards" v-if="cards.length > 1">
      <card
        v-for="(card, index) in cards" :key="index"
        :title="card.title"
        :icon="card.icon"
        :value="card.value.toString()"
      >
    </card>
    <button class="calc" @click="subnetsCalc()">calcular</button>
  </div>
  <section class="subnets-list" v-if="hasSubnets">
    <VTable
      title="Lista de Subredes"
      :list="subnetListSubnet"
      @modalShow="showModal = true"
      :calc="hostCalc"
      :pre="prefixSubnet"
    >
  </VTable>
</section>
  <modal v-if="showModal" @close="showModal = false" :hosts="hostsList"
    :broad="broadcastAdrrSubnet"
  >
    <h3>custom header</h3>
  </modal>
</div>
</template>
```

Ejecución del programa

El programa fue desplegado en web, se puede ver dicho programa ingresando a esta liga: <https://ipcalculator.netlify.app/>

De entrada, veremos la siguiente interfaz:

The screenshot shows the 'CALCULADORA IP' web application. At the top, there are three tabs: 'Subnets' (selected), 'Hosts', and 'Prefix'. Below the tabs, there are two input fields: 'Dirección IP' (containing '192.168.0.0') and 'No. Subredes' (containing '0'). At the bottom, there is a footer that says 'With ❤️ by SiusuTeam'.

Para comenzar a ver el funcionamiento tenemos que ingresar la dirección ip válida y el elemento para hacer los cálculos según la pestaña donde estemos (subredes, hosts, prefijo)

Primeramente, probaremos con las subredes:

The screenshot shows the 'CALCULADORA IP' web application with the 'Subnets' tab selected. The 'Dirección IP' field contains '192.168.0.0' and the 'No. Subredes' field contains '500'. Below the input fields, there are five result boxes: 'CLASE DE RED: Clase A', 'MÁSCARA DE RED: 255.0.0.0', 'SUBREDES: 512', 'HOSTS: 32766', and 'NUEVA MÁSCARA: 255.255.128.0'. A large 'CALCULAR' button is centered below these results. At the bottom, there is a footer that says 'With ❤️ by SiusuTeam'.

Podemos ver que se ingreso la dirección 10.0.0.0 y un total de 500 subredes requeridas por el usuario, al ingresar estos datos y que sean válidos, se desplegará en tarjetas la información de la red, además de un botón para calcular las subredes.

Al dar en el botón de calcular se desplegará la lista con el total de subredes calculadas:

LISTA DE SUBREDES	
ID	DIRECCIÓN
1	10.0.0.0/17
2	10.0.128.0/17
3	10.1.0.0/17
4	10.1.128.0/17
5	10.2.0.0/17
6	10.2.128.0/17
7	10.3.0.0/17
8	10.3.128.0/17
9	10.4.0.0/17
10	10.4.128.0/17
11	10.5.0.0/17
12	10.5.128.0/17
13	10.6.0.0/17
14	10.6.128.0/17
15	10.7.0.0/17
16	10.7.128.0/17
17	10.8.0.0/17
18	10.8.128.0/17
19	10.9.0.0/17

Da clic en una dirección para saber sus direcciones de host

With ❤ by SiusuTeam

Para obtener las direcciones de host para cada subred, tenemos que dar clic en la dirección que deseemos como indica el mensaje en azul.

DIR. DE BROADCAST: 10.0.255.255

X

LISTA DE HOSTS

ID	DIRECCIÓN
1	10.0.128.1/
2	10.0.128.2/
3	10.0.128.3/
4	10.0.128.4/
5	10.0.128.5/
6	10.0.128.6/
7	10.0.128.7/
8	10.0.128.8/
9	10.0.128.9/
10	10.0.128.10/
11	10.0.128.11/
12	10.0.128.12/
13	10.0.128.13/
14	10.0.128.14/
15	10.0.128.15/
16	10.0.128.16/
17	10.0.128.17/
18	10.0.128.18/
19	10.0.128.19/
20	10.0.128.20/

Al calcular las direcciones de host, se nos desplegará un modal como se ve en la imagen anterior, este incluye las direcciones de host para la subnet elegida.

Para los cálculos a partir del número de host y prefijo son similares, por lo que solo se mostrarán capturas con los resultados:

A partir de host

CALCULADORA IP

Subnets

Hosts

Prefix

Dirección IP
192.128.10.0

No. Hosts
50

CLASE DE RED
Clase C

MÁSCARA DE RED
255.255.255.0

SUBREDES
4

HOSTS
62

NUEVA MÁSCARA
255.255.255.192

CALCULAR

With ❤️ by SiuuuTeam

CLASE DE RED
Clase C

MÁSCARA DE RED
255.255.255.0

SUBREDES
4

HOSTS
62

NUEVA MÁSCARA
255.255.255.192

CALCULAR

LISTA DE SUBREDES

ID	DIRECCIÓN
1	192.128.10.0/26
2	192.128.10.64/26
3	192.128.10.128/26
4	192.128.10.192/26

With ❤️ by SiuuuTeam

DIR. DE BROADCAST: 192.128.10.127

X

LISTA DE HOSTS

ID	DIRECCIÓN
1	192.128.10.65/
2	192.128.10.66/
3	192.128.10.67/
4	192.128.10.68/
5	192.128.10.69/
6	192.128.10.70/
7	192.128.10.71/
8	192.128.10.72/
9	192.128.10.73/
10	192.128.10.74/
11	192.128.10.75/
12	192.128.10.76/
13	192.128.10.77/
14	192.128.10.78/
15	192.128.10.79/
16	192.128.10.80/
17	192.128.10.81/
18	192.128.10.82/
19	192.128.10.83/

A partir del prefijo

CALCULADORA IP


Subnets


Hosts


Prefix


Dirección IP
129.10.0.0


Prefijo
20

 CLASE DE RED
Clase A

 MÁSCARA DE RED
255.0.0.0

 SUBREDES
4096

 HOSTS
4094

 NUEVA MÁSCARA
255.255.240.0

CALCULAR

With ❤️ by SiuuuTeam

LISTA DE SUBREDES

ID	DIRECCIÓN
1	129.0.0.0/20
2	129.0.16.0/20
3	129.0.32.0/20
4	129.0.48.0/20
5	129.0.64.0/20
6	129.0.80.0/20
7	129.0.96.0/20
8	129.0.112.0/20
9	129.0.128.0/20
10	129.0.144.0/20
11	129.0.160.0/20
12	129.0.176.0/20
13	129.0.192.0/20
14	129.0.208.0/20
15	129.0.224.0/20
16	129.0.240.0/20
17	129.1.0.0/20
18	129.
19	129.

Da clic en una dirección para saber sus direcciones de host

With ❤️ by SiuuuTeam

DIR. DE BROADCAST: 120.0.143.255

X

LISTA DE HOSTS

ID	DIRECCIÓN
1	120.0.128.1/
2	120.0.128.2/
3	120.0.128.3/
4	120.0.128.4/
5	120.0.128.5/
6	120.0.128.6/
7	120.0.128.7/
8	120.0.128.8/
9	120.0.128.9/
10	120.0.128.10/
11	120.0.128.11/
12	120.0.128.12/
13	120.0.128.13/
14	120.0.128.14/
15	120.0.128.15/
16	120.0.128.16/
17	120.0.128.17/
18	120.0.128.18/
19	120.0.128.19/

Conclusiones

Fischer Salazar César Eduardo

El subneteo me parece una herramienta bastante útil la cual nos permite fraccionar una dirección IP a redes más pequeñas para trabajar de manera individual y más eficiente, puede parecer algo complicado por las consideraciones que se deben tener dependiendo de su clasificación, además de que de hacerse a mano este puede tener el factor de error humano que haría que no se llevara a cabo un buen funcionamiento de estas

Siendo mi primer contacto con el Subneteo de redes tradicional y el VLSM me costó bastante el entender el manejo de un número grande de subredes, así como el poder identificar el prefijo y la nueva máscara de subred correspondiente a esta división, pero la aplicación de estos métodos en la calculadora me ayudó a comprender de mejor manera el cómo estas se dividen en sus redes, aunque no del todo porque hay unos aspectos que aún me fallan.

Me pareció interesante la elaboración de este proyecto pues me permitió conocer un aspecto que desconocía completamente sobre las redes y que en un futuro podría serme útil en la vida laboral.

López García José Eduardo

Dentro de mi experiencia en vocacional con los métodos de cálculo de IP me ha ayudado a reforzar paso a paso todos los puntos que se han tocado dentro de la materia con respecto a este tema, ya que realizar dichos métodos ayuda a comprender qué es lo que se pretende realizar, sin embargo, puede ser un trabajo bastante elaborado cuando se intente manejar cantidades más grandes de subred y se necesita de alguna manera querer optimizar el tiempo, y en una de esas es posible caer en un error que nos cueste todo el esquema y puedan presentarse diversas fallas.

Esta práctica nos dio una apertura mucho más amplia a cómo es que se realizan estas herramientas, que de verdad han podido ser muy útiles y han podido generar un apoyo inmenso a estudiantes y administradores de red; entendimos muy claro qué métodos, funciones y características se emplean para determinar las direcciones IP, su clase, la cantidad de host y subred, y las submáscaras que lleva cada una, caímos en cuenta que suele ser un trabajo complejo realizar la programación hasta de una de las partes más sencilla dentro del cálculo.

Sin duda, haber sentado las bases de lo aprendido en el cálculo de IP ha funcionado para cumplir con el objetivo de la calculadora, ya que sirvió como una guía de que se estaba implementando un buen trabajo con el programa, que desplegaba la información correcta de nuestras IP's, y que se estaban respetando los límites entre cada subred y host; he utilizado como 2 o 3 calculadoras online en mi etapa de la preparatoria para realizar comprobaciones de mis resultados, estoy convencido que este trabajo será de mucha ayuda para diferentes personas que estén interesadas en estos temas.

Meza Vargas Brandon David

El Subneteo de una red IP es un trabajo que muchas veces es tardado si se hace de manera manual sin requerir a un software, es bueno saberlo, pero en un escenario práctico no es lo mejor hacer esto a mano, pues requiere de mucho tiempo y si te equivocas en un cálculo puede echar a perder el Subneteo pues obtendrás datos erróneos que son críticos, por esta razón fue el desarrollo de nuestro pequeño proyecto.

El aterrizar los conocimientos que hemos adquirido sobre Subneteo en las clases de redes en un software que te permita hacer cálculos con una dirección IP ingresada por el usuario además de un número de subredes deseado, hosts o a partir del prefijo, nos dejó mucha experiencia en este tema del Subneteo fortaleciendo así el conocimiento previo adquirido.

Una vez más confirmamos la importancia que tiene el Subneteo en las redes actualmente, pues es una práctica que se sigue usando y se seguirá usando por un largo tiempo. Desarrollar un software que permita hacer esto de forma automáticamente resulta crucial en estos tiempos por motivos anteriormente mencionados.

Un proyecto muy interesante que incentiva a los estudiantes a conocer a fondo el tema y dar solución implementando lenguajes de programación de su preferencia para no sentirse presionado, además que hace que los estudiantes trabajen su lógica para dar la solución correcta.