Ministério da Educação
UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
Campus Campo Mourão
Curso de Bacharelado em Ciência da Computação

TRABALHO DE REDES DE COMPUTADORES 1

ATIVIDADE PRÁTICA

Programa de Cálculo de Rede

Enzo Dornelles Italiano[1]
Henrique Souza Marcuzzo[2]

CAMPO MOURÃO
DEZ/2019

[1] enzoitaliano@alunos.utfpr.edu.br
[2] henriquemarcuzzo@alunos.utfpr.edu.br

SUMÁRIO

1. INTRODUÇÃO

Este trabalho tem como objetivo desenvolver um software que realize cálculos básicos que são utilizados na implementação de redes de computadores, assim como fixar melhor os conhecimentos adquiridos durante as aulas. O relatório tem o intuito de documentar o desenvolvimento e funcionamento do software a ser implementado.

2. LINGUAGEM UTILIZADA

A linguagem escolhida para a implementação foi a Python, na versão 3.8.0 por conta de ser uma linguagem simples e fácil por conta de ser de alto nível, contendo uma sintaxe simples e com boa visualização, por conta disso o aprendizado da mesma é muito mais fácil e rápido. Além de ser multiplataforma, ou seja, funcional em muitos sistemas operacionais.

3. A TECNOLOGIA JSON

JavaScript Object Notation (JSON) é um modelo para armazenamento e transmissão de informações no formato texto e vem cada vez mais nos últimos anos sendo utilizado em sistemas web, substituindo o XML que tem a mesma finalidade, por conta de conseguir estruturar informações de forma mais compacta, tornando mais rápida a separação de tais informações.

Um objeto JSON simples por exemplo, tem a seguinte configuração:

```
{
      "ObjetoPai":"valor"
}
```

Um mais complexo teria a seguinte configuração com objetos-pai e objetos-filho:

```
{
      "ObjetoPai":{
            "ObjetoFilho":"valor"
      }
}
```

4. ENDEREÇAMENTO IP

O endereço IP é uma sequência de 32 bits dividido em 4 grupos de 8 bits que identifica um dispositivo na rede. Sendo ele dividido em duas partes, a primeira para identificar a rede e a segunda para identificar o host dentro da rede, a quantidade de octetos para cada uma é diferente dependendo da classe que pode ser A, B ou C atualmente.

Na classe A, apenas o primeiro octeto é referente a rede e os outros três ao host.

Na classe B, a divisão é igual, dois octetos para rede e dois para o host.

A classe C é o inverso da classe A, com três octetos para rede e um para host.

	255	255	255	255
Classe A	Rede	Host	Host	Host
Classe B	Rede	Rede	Host	Host
Classe C	Rede	Rede	Rede	Host

Tabela 1: Bits de rede e de host.

Ao configurar uma rede, a escolha de qual classe usar vai depender de quantas máquinas estarão na rede, caso forem menos de 254 a classe C é a mais recomendada pois é necessário configurar apenas o último octeto, caso sejam mais, a classe B é uma boa opção. É muito difícil encontrar redes que necessitem de uma classe A.

Todos os endereços IP disponíveis já possuem dono e portanto existem regras para endereços TCP/IP válidos:

Classe A	1 a 126	0 a 255	0 a 255	0 a 255
Classe B	128 a 191	0 a 225	0 a 255	0 a 255
Classe C	192 a 223	0 a 225	0 a 225	1 a 254

Tabela 2: Valores mínimos e máximos para cada octeto.

Para a classe A e B, os espaços de host não podem ser todos 0 e nem todos 255.

5. O CÓDIGO

O código foi colocado dentro de uma função main para maior compreensão.

A primeira função a ser chamada é a init onde são pegos o endereço Ip e a máscara de rede contidos no arquivo JSON de entrada e atribuídos à variáveis para manipulação pelo código.

```
def init():
    configFile = sys.argv[1]
    inputFile = open(configFile, 'r')
    datas = json.load(inputFile)

    ipAddr = datas['ipAddr']
    netMask = datas['netMask']
    return ipAddr, netMask
```

Continuando a arrumação para manipular os dados na função convertToInteger são removidos os pontos que separam os octetos e os números que vêm como string são transformados em inteiros.

```
def convertToInteger(ipAddr, netMask):
    ipAddr = ipAddr.split(".")
    netMask = netMask.split(".")

    for i in range(len(ipAddr)):
        ipAddr[i] = int(ipAddr[i])

    for i in range(len(netMask)):
        netMask[i] = int(netMask[i])
    return ipAddr, netMask
```

Depois é necessário convertê-los para binário utilizando a função convertToBinary e a função própria do python bin( ) e tornar comparações e cálculos mais fáceis e relacionados ao conteúdo passado em sala.

```
def convertToBinary(ipAddr, netMask):
    auxIP = ipAddr
    auxNet = netMask

    for i in range(4):
        auxIP[i] = bin(auxIP[i])
        auxNet[i] = bin(auxNet[i])
    return auxIP, auxNet
```

A próxima função chama verifica se o IP e a máscara de redes são válidas, visto que nenhum valor pode ser menor que 0 ou maior do que 255 e na máscara, depois que algum zero aparecer não pode haver mais nenhum um.

```
def isValid(ipAddrI, netMaskI, ipAddrB, netMaskB):
    flag_zero = 0
    for i in range(4):
        if (ipAddrI[i] > 255 or ipAddrI[i] < 0):
            print("O IP digitado é inválido!\n")
            exit()
        if (netMaskI[i] > 255 or netMaskI[i] < 0):
            print("A mascara de IP digitada é inválida!\n")
            exit()
    for i in range(len(netMaskB)):
        for j in range(2, len(netMaskB[i])):
            if netMaskB[i][j] is '1' and flag_zero is 1:
                print("A mascara de IP digitada é inválida!\n")
                exit()
            if netMaskB[i][j] is '0':
                flag_zero = 1
```

A função host_NetWork tem a finalidade de identificar quantos bits são relativos a rede e quantos são para host, no total de 32. Essa identificação é feita utilizando a máscara de rede e onde encontrarmos 1 significa que está sendo utilizado para rede. Por exemplo 11111111.111111111.11111111.00000000 possui 28 bits para rede e 8 para host, com um total de hosts máximo de 254.

```
def host_NetWork(netMask):
    netID = 0
    hostID = 0
    flag = 0

    for i in range(len(netMask)):
        for j in range(2, len(netMask[i])):
            if netMask[i][j] is '0':
                flag = 1
            if flag == 0:
                netID += 1

    hostID = 32 - netID
    return netID, hostID
```

A classe é identificada verificando os valores utilizados na Tabela 2. Outras classes não explicadas antes também são verificadas, D e E, não são utilizadas atualmente pois são uma predefinição para expansões futuras.

```
def isClass(ipAddr):
    if (ipAddr[0] >= 0 and ipAddr[0] <= 127):
        classe = "A"
    elif (ipAddr[0] >= 128 and ipAddr[0] <= 191):
        classe = "B"
    elif (ipAddr[0] >= 192 and ipAddr[0] <= 223):
        classe = "C"
    elif (ipAddr[0] >= 224 and ipAddr[0] <= 239):
        classe = "D"
    elif (ipAddr[0] >= 240 and ipAddr[0] <= 255):
        classe = "E"
    return classe
```

Para encontrar IP da rede foi pego cada octeto do vetor que estava em binário e para a quantidade de bits de rede foi se adicionando 1 ou 0 de acordo com o vetor e adicionando 0 ao final para completar os 32 bits em uma string auxiliar, feito isso cada octeto foi convertido de binário para decimal.

```
def findIpNetwork(ipAddrB, netID):
    IpNetwork = []
    for i in range(4):
        aux = ''
        for j in range(10):
            if (netID != 0):
                if (j < len(ipAddrB[i])):
                    aux += ipAddrB[i][j]
                if (j > 1):
                    netID -= 1
            else:
                if (j > 1):
                    aux += '0'
        IpNetwork.append(int(aux, 2))
    return IpNetwork
```

Para encontrar o IP de broadcast o método utilizado foi o mesmo, tendo como única diferença a adição de uns ao final em vez de zeros.

```
def findIpBroadcast(ipAddrB, netID):
    IpBroadcast = []
    for i in range(4):
        aux = ''
        for j in range(10):
            if (netID != 0):
                if (j < len(ipAddrB[i])):
                    aux += ipAddrB[i][j]
                if (j > 1):
                    netID -= 1
            else:
                if (j > 1):
                    aux += '1'
        IpBroadcast.append(int(aux, 2))
    return IpBroadcast
```

Utilizando o IP da rede e o IP de broadcast foi calculada a faixa de hosts válidos, sendo o IP de rede com o final mais um, até o IP de broadcast com o final menos um.

```
def findIpValid(ipNetwork, ipBroadcast):

    ipNetwork[3] = ipNetwork[3] + 1
    ipBroadcast[3] = ipBroadcast[3] - 1
    return ipNetwork, ipBroadcast
```

Para identificar o status do IP foi utilizada a seguinte tabela.

Bloco de Endereços	Descrição
0.0.0.0/8	Rede corrente (só funciona como endereço de origem)
10.0.0.0/8	Rede Privada
14.0.0.0/8	Rede Pública
39.0.0.0/8	Reservado
127.0.0.0/8	Localhost
128.0.0.0/16	Reservado (IANA)
169.254.0.0/16	Zeroconf
172.16.0.0/12	Rede privada
191.255.0.0/16	Reservado (IANA)
192.0.2.0/24	Documentação
192.88.99.0/24	IPv6 para IPv4
192.168.0.0/16	Rede Privada
198.18.0.0/15	Teste de benchmark de redes
223.255.255.0/24	Reservado
224.0.0.0/4	Multicasts (Classe D)
240.0.0.0/4	Reservado (Classe E)
255.255.255.255	Broadcast

Tabela 3: Blocos de Endereços Reservados

E por meio de condicionais se verifica qual é o status da rede

```
def networkState(ipAddr):

    if (ipAddr[0] == 0):
        status = "Rede corrente"
    elif ((ipAddr[0] == 10) or (ipAddr[0] == 172 and ipAddr[1] == 16) or
(ipAddr[0] == 192 and ipAddr[1] == 168)):
        status = "Rede privada"
    elif (ipAddr[0] == 14):
        status = "Rede pública"
    elif ((ipAddr[0] == 39) or (ipAddr[0] == 128 and ipAddr[1] == 0) or
```

```
(ipAddr[0] == 191 and ipAddr[1] == 255)
            or (ipAddr[0] == 223 and ipAddr[1] == 255 and ipAddr[2] ==
255) or (ipAddr[0] == 240)):
        status = "Reservado"
    elif (ipAddr[0] == 127):
        status = "Localhost"
    elif (ipAddr[0] == 169 and ipAddr[1] == 254):
        status = "Localhost"
    elif (ipAddr[0] == 192 and ipAddr[1] == 0 and ipAddr[2] == 2):
        status = "Documentação"
    elif (ipAddr[0] == 192 and ipAddr[1] == 88 and ipAddr[2] == 99):
        status = "IPv6 para IPv4"
    elif (ipAddr[0] == 198 and ipAddr[1] == 18):
        status = "Teste de benchmark de redes"
    elif (ipAddr[0] == 224):
        status = "Multicasts"
    elif (ipAddr[0] == 255 and ipAddr[1] == 255 and ipAddr[2] == 255 and
ipAddr[3] == 255):
        status = "Broadcast"
    else:
        status = "Ip não reservado"
    return status
```

Para finalizar, a função output é chamada, a fim de colocar todas as informações calculadas em um arquivo JSON de saída. Este arquivo pode ter sido passado por linha de comando ou não.

```
def output(netID_hostID, classe, ipNetwork, ipBroadcast, ipValid,
status):

    results = {
        "Bits_de_rede": netID_hostID[0],
        "Bits_de_host": netID_hostID[1],
        "Hosts_na_rede": (2 ** netID_hostID[1]) - 2,
        "Classe_da_Rede": classe,
        "Ip_da_rede": str(ipNetwork[0]) + "." + str(ipNetwork[1]) + "." +
str(ipNetwork[2]) + "." + str(ipNetwork[3]),
        "Ip_de_broadcast": str(ipBroadcast[0]) + "." +
str(ipBroadcast[1]) + "." + str(ipBroadcast[2]) + "." +
str(ipBroadcast[3]),
        "Ip_valido_inicial": str(ipValid[0][0]) + "." +
str(ipValid[0][1]) + "." + str(ipValid[0][2]) + "." + str(ipValid[0][3]),
        "Ip_valido_final": str(ipValid[1][0]) + "." + str(ipValid[1][1])
+ "." + str(ipValid[1][2]) + "." + str(ipValid[1][3]),
        "Status_do_IP": status
```

```
    }

    if len(sys.argv[1:]) > 1:
        outputFile = sys.argv[2]
    else:
        outputFile = "output.json"

    with open(outputFile, 'w') as file:
            json.dump(results, file, indent = 4)
```

6. COMO EXECUTAR

Para executar, são necessários dois arquivos, o main.py com todo o código que foi descrito acima e o input.json que é o arquivo de entrada que contém o endereço de IP e a máscara de rede.

A estrutura do arquivo de entrada deve ser a seguinte:

```
{
      "ipAddr": "XXX.XXX.XXX.XXX",
      "netMask": "YYY.YYY.YYY.YYY"
}
```

Sendo X o endereço de IP e Y a máscara de rede.

6.1. Exemplo

Considerando o input.json como:

```
{
      "ipAddr": "192.168.0.1",
      "netMask": "255.255.255.0"
}
```

Enviamos o seguinte comando para o terminal:

```
python3 main.py input.json output.json
```

Devemos ter a seguinte saída do terminal:

```
Bits de rede: 24
Bits de host: 8
Hosts na rede: 254
Classe: C
```

```
Ip da rede: 192.168.0.0
Ip de broadcast: 192.168.0.255
Faixa de hosts válidos: 192.168.0.1 - 192.168.0.254
Status do IP: Rede privada
```

E o arquivo output.json deve ter a seguinte saída:

```
{
    "Bits_de_rede": 24,
    "Bits_de_host": 8,
    "Hosts_na_rede": 254,
    "Classe_da_Rede": "C",
    "Ip_da_rede": "192.168.0.0",
    "Ip_de_broadcast": "192.168.0.255",
    "Ip_valido_inicial": "192.168.0.1",
    "Ip_valido_final": "192.168.0.254",
    "Status_do_IP": "Rede privada"
}
```

7. CONCLUSÃO

O objetivo do trabalho foi concluído depois de mostrar no terminal e guardar em json todas as informações requisitadas no projeto que são necessárias para configurar uma rede.

Alguns obstáculos apareceram durante o percurso por não saber exatamente como funciona a passagem de parâmetros para funções em python, assim como o uso das funções bin( ) e int( ) que são próprias da linguagem. Também foi uma tarefa mais difícil no projeto pegar os dados do arquivo json mesmo com sua biblioteca específica que tem integração com a linguagem.

No entanto, em relação às transformações em si, tudo foi mais fácil depois de os valores terem sido convertidos para valores binários, o que simplificou bastante os cálculos.

8. REFERÊNCIAS

MORIMOTO, Carlos E. Redes e Servidores Linux. 2.ed. São Paulo: GDH Press e Sul Editores, 2006.

O que é JSON? Para que serve e como funciona? (2014). Disponível em: <https://pt.stackoverflow.com/questions/4042/o-que-%C3%A9-json-para-que-serve-e-como-funciona>. Acessado em dezembro de 2019.

Endereço IP (2019). Disponível em: <https://pt.wikipedia.org/wiki/Endere%C3%A7o_IP>. Acessado em dezembro de 2019.

Redes – Classes de endereços IP, sabe quais são? (2011). Disponível em: <https://pplware.sapo.pt/tutoriais/networking/classes-de-endereos-ip-sabe-quais-so/>. Acessado em dezembro de 2019.