Lezione 17 - Programmazione sugli IP

IP address

Jupyter Notebook per mostrare l'uso di base del modulo `ipaddress`.
Fate riferimento a questo tutorial per maggiori informazioni.
Per prima cosa importiamo tutto quello che ci servirà.

```
import ipaddress
```

Il modulo `ipaddress` supporta tre tipi di oggetto:

- indirizzi (degli host): solo indirizzo IP
- indirizzi delle reti: indirizzo di rete associato ad una maschera/lunghezza del prefisso (che definisce un insieme di indirizzi che hanno lo stesso prefisso)
- indirizzi delle interfacce: ibrido tra i due, in quanto combina le informazioni sulla lunghezza del prefisso con una parte di host non zero

```
addr1 = ipaddress.ip_address('142.251.209.3') #factory in
grado di riconoscere notazione IPv4 e IPv6
addr2 = ipaddress.IPv4Address('142.251.209.3')
#costruttore per gli indirizzi IPv4
print(repr(addr1))
print(repr(addr2))

#Output
#IPv4Address('142.251.209.3')
#IPv4Address('142.251.209.3')*/
```

Gli indirizzi possono essere convertiti in numeri

```
int(ipaddress.IPv4Address('142.251.209.3'))

#Output
#'142.251.209.3'
```

Usa una `f-string` dove chiedo di formattarlo in maniera numerica (`n`: binario nel caso IPv4 e esadecimale nel caso IPv6) con separatore (`_`). Uso `replace` per usare invece lo spazio come separatore.

```
f"{ipaddress.IPv4Address('142.251.209.3'):_n}".replace('_
',' ')

#Output
#'1000 1110 1111 1011 1101 0001 0000 0011'
```

Possiamo confrontare due indirizzi secondo l'ordine consueto. Il confronto come stringa potrebbe invece portare a risultati sbagliati (si noti, che per esempio '100' è minore di '2')

```
a = ipaddress.IPv4Address('142.251.209.3')
b = ipaddress.IPv4Address('142.251.209.4')
a<b

#Output True
```

Possiamo sommare o sottrarre numeri da un indirizzo.

```
a = ipaddress.IPv4Address('142.251.209.3')
a + 1
```

```
#Output  IPv4Address('142.251.209.4')
```

Considerando la sua rappresentazione binaria, abbiamo avuto modo di vedere che l'indirizzo IP `142.251.209.3` è di classe B, quindi la parte di rete e quella di host sono entrambe lunghe 16 bit.
Possiamo costruire un `interface address` per rappresentare l'indirizzo IP insieme con le informazioni sulla *subnet mask*

```
addr1 = ipaddress.ip_address('142.251.209.3/16') #factory
in grado di riconoscere notazione IPv4 e IPv6
addr2 = ipaddress.IPv4Address('142.251.209.3/16')
#costruttore per gli indirizzi IPv4
print(repr(addr1))
print(repr(addr2))


#Output
#IPv4Interface('142.251.209.3/16')
IPv4Interface('142.251.209.3/16')
```

Dalla interfaccia si possono ottenere varie informazioni

- *Subnet mask*: indica quali bit appartengono alla parte di rete

```
addr1.netmask

#Output  IPv4Address('255.255.0.0')
```

- *Host mask*: indica quali bit appartengono alla parte dell'host

```
addr1.hostmask

#Output  IPv4Address('0.0.255.255')
```

- Indirizzo (dell'host) senza informazioni sulla sottorete

```
addr1.ip

#Output  IPv4Address('142.251.209.3')
```

- Indirizzo della rete

```
addr1.network

#Output  IPv4Address('142.251.0.0/16')
```

Anche per gli indirizzi di rete ho due modi per costruirli

```
print(repr(ipaddress.ip_address('142.251.209.3/16')))
print(repr(ipaddress.IPv4Network('142.251.209.3/16')))

#Output
#IPv4Interface('142.251.209.3/16')
IPv4Interface('142.251.209.3/16')
```

- Lunghezza del prefisso di una rete

```
ipaddress.IPv4Network('142.251.209.3/16').prefixlen

#Output  16
```

- Sua netmask

```
ipaddress.IPv4Network('142.251.209.3/16').netmask

#Output  IPv4Address('255.255.0.0')
```

- E hostmask

```
ipaddress.IPv4Network('142.251.209.3/16').hostmask

#Output  IPv4Address('0.0.255.255')
```

- Posso sapere l'indirizzo di broadcast di una rete

```
ipaddress.IPv4Network('142.251.209.3/16').broadcast_addres
s

#Output  IPv4Address('142.251.255.255')
```

- Posso sapere il numero di indirizzi in una rete(comprensivi dell'indirizzo della rete e di quello di broadcast)

```
ipaddress.IPv4Network('142.251.209.3/16').num_address

#Output  65536
```

- Posso anche enumerare sugli indirizzi in generale e se quelli degli host

```
import itertools
for a in
```

```
itertools.islice(ipaddress.IPv4Network('142.251.0.0/16'),
10):
        print(a)
print('...')

#Output
#142.251.0.0
#142.251.0.1
#142.251.0.2
#142.251.0.3
#142.251.0.4
#142.251.0.5
#142.251.0.6
#142.251.0.7
#142.251.0.8
#142.251.0.9
# ...
```

```
import itertools
for a in
itertools.islice(ipaddress.IPv4Network('142.251.0.0/16').h
ost(), 10):
        print(a)
print('...')

#Output
#142.251.0.1
#142.251.0.2
#142.251.0.3
#142.251.0.4
#142.251.0.5
#142.251.0.6
#142.251.0.7
#142.251.0.8
#142.251.0.9
```

```
#142.251.0.10
# ...
```

Longest prefix match

Per prima cosa importiamo tutto quello che ci servirà.

```
import ipaddress
```

Tabella di inoltro (basato sulla destinazione)

```
forwarding_table=[
        ("142.251.200.0/24", 0),
        ("142.251.192.0/18", 1),
        ("142.251.0.0/16", 2),
        ("0.0.0.0/0", 3)
]
```

Indirizzo di destinazione del pacchetto da instradare

```
destination = '142.251.209.3'
```

Itero sulle voci della tabella di inoltro, confrontando per ciascuna di esse l'indirizzo di destinazione con il prefisso di rete indicato. Prendo l'uscita cui corrisponde il prefisso più lungo.

```
output_link = None
matched_prefix_lenght = 0
desaddr = ipaddress.ip_address(destination)
for prefix, link in forwarding_table:
        netaddr = ipaddress.ip_network(prefix)
        print(f'Check IP address {destaddr} against route
```

```
{netaddr} via link {link}...', end='')
        if destaddr in netaddr:
                print('ok', end='')
                if netaddr.prefixlen >=
matched_prefix_length:
                        matched_prefic_length =
netaddr.prefixlen
                        output_link = link
                        print(' [select new link]')
                else:
                        print()
        else:
        print('not ok')

print(f'The packet destinated to {destaddr} will be
foreward throught the output link {outputl_link}')


#Output
#Check IP address 142.251.209.3 against route
142.251.200.0/24 via link 0...not ok
#Check IP address 142.251.209.3 against route
142.251.192.0/18 via link 1...ok [select new link]
#Check IP address 142.251.209.3 against route
142.251.0.0/16 via link 2...ok
#Check IP address 142.251.209.3 against route 0.0.0.0/0
via link 3...ok
#The packet destinated to 142.251.209.3 will be forwared
through the output link 1
```

Quello proposto è un algoritmo molto semplice, finalizzato alla semplicità di comprensione piuttosto che alle prestazioni.

Subnetting

Per prima cosa importiamo tutto quello che ci servirà.

```
import ipaddress
```

Indirizzo di rete da estendere e numero di bit di cui estendere

```
network_address = ipaddress.IPv4Network('131.175.0.0/21')
diff = 1

for subnet in
network_address.subnets(prefixlen_diff=diff):
        print(subnet)


#Output
#131.175.0.0/22
#131.175.4.0/22
```

Si può altrimenti dare la nuova lunghezza del prefisso

```
network_address = ipaddress.IPv4Network('131.175.0.0/21')
prefixlen = 22

for subnet in
network_address.subnets(new_prefix=prefixlen):
        print(subnet)


#Output
#131.175.0.0/22
#131.175.4.0/22
```

Oppure in maniera più compatta

```
[x for x in network_address.subnets(new_prefix=prefixlen)]

#Output
#[IPv4Network('131.175.0.0/22'),
IPv4Network('131.175.4.0/22')]
```

Esiste anche l'operazione inversa che accorcia un prefisso di rete

```
subnet_address = ipaddress.IPv4Address('131.175.4.0/22')
print(repr(subnet_address.supernet(prefixlen_diff = 1)))
print(repr(subnet_address.supernet(new_prefix = 21))

#Output
#IPv4Network('131.175.0.0/21'),
IPv4Network('131.175.4.0/21')
```

Possiamo anche chiedere se una rete sia una sottorete di un'altra

```
net1 = ipaddress.IPv4Network('131.175.0.0/21')
net2 = ipaddress.IPv4Network('131.175.4.0/22')
print(net1.subnet_of(net2))

#Output True
```