Telekommunikációs Hálózatok

2. gyakorlat

JSON, subprocess

PYTHON ALAPOK II.

A struct modul

– A struct.**pack**(*fmt*, *v1*, *v2*, …): egy sztringgel tér vissza, amely az adott *fmt* formátumnak megfelelően csomagolja be a bemenetként kapott értéke(ke)t (binárissá alakítja)
  - pl. formátumoknál a 'b' (előjeles) char C-típust (1-bájtos egész),
  - a '4sL' 4 méretű char tömböt és egy (előjel nélküli) long C-típust (4-bájtos egész) jelöl
  - Továbbiak: https://docs.python.org/3/library/struct.html#format-characters

Struktúra létrehozása

- Binárissá alakítjuk az adatot
  - A Struct konstruktorában a formátumot adjuk meg, - hasonlóan az előbbihez, - amely alapján írja/olvassa a bináris adatot
  - A *-operátor az alábbi esetben úgy fog viselkedni, mintha ','-vel elválasztva felsoroltuk volna a *values* elemeit

```
import struct
values = (1, b'ab', 2.7)
packer = struct.Struct('I 2s f')                #Int, char[2], float
packed_data = packer.pack(*values)
```

- Visszaalakítjuk a kapott üzenetet

```
import struct
unpacker = struct.Struct('I 2s f')
unpacked_data = unpacker.unpack(data)
```

- megj.: integer 1 – 4 byte, sztringként 1 byte, azaz hatékonyabb char tömbként átküldeni.

Struktúra jellemzői

ASCII karaktereket tartalmazhat

- Mire kell figyelni?
  - A Struct formátumnál az „Xs” (pl. „2s”) X db. bájtból álló **bájtliterált** jelent (pl. **b'abc'**) és NEM Unicode karakterláncot (pl. 'abc')!

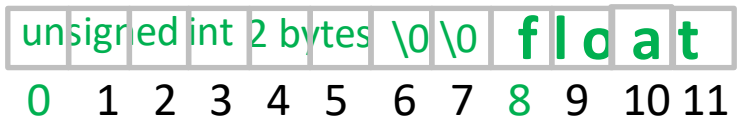
```
import struct
values = (1, 'ab', 2.7)
packer = struct.Struct('I 2s f')
packed_data = packer.pack(*values)
# HIBA: struct.error: argument for 's' must be a bytes object
# JÓ megoldás:
values = (1, b'ab', 2.7) # vagy values = (1, 'ab'.encode(), 2.7)
…
```

Struktúra jellemzői

- Mire kell figyelni?
  - Az alábbi parancsokkal lehet ellenőrizni, hogy hány bájtot használ egy struktúra:

```
import struct
packer = struct.Struct('I 2s f')
print(struct.calcsize('I 2s f '))
print(packer.size)
# 12
```

  - I: unsigned int size = 4, 2s: 2 bytes, f: float size = 4, 4+2+4 ≠ **12** ???
  - Alapból a struct natív igazítást (native alignment) használ, emiatt egy int/float-ot úgy kell igazítani, hogy a kezdő pozíciója 4-gyel osztható legyen



Fájlkezelés

- A fájlkezelés során lehet állítani (seek), hogy mi legyen a fájlobj.-nak az aktuális pozíciója:

```
with open('alma.txt', 'r') as f:
# tegyük fel, hogy az alma.txt tartalma:
# 1. sor
# 2. sor
# 3. sor
  sor = f.readline()
  print('jelenlegi sor', sor) # jelenlegi sor 1. sor
  sor = f.readline()
  print('jelenlegi sor', sor) # jelenlegi sor 2. sor
  f.seek(0, 0)                # f.seek(offset, whence)
  sor = f.readline()
  print('jelenlegi sor', sor) # jelenlegi sor 1. sor
```

- offset: olvasás/írás mutató pozíciója a fájlban
- (whence: alapért. 0: abszolút poz., 1: relatív aktuális poz.-hoz, 2: rel. a fájl végéhez)

Fájlkezelés

- Bináris fájlt is lehet írni/olvasni
  - és abban is működik a seek fv.

```
import struct
packer = struct.Struct('i3si')
values = (2020, b'jan', 18)
packed_date1 = packer.pack(*values)
values = (2020, b'nov', 12)
packed_date2 = packer.pack(*values)
with open('dates.bin', 'wb') as f:
  f.write(packed_date1)
  f.write(packed_date2)

with open('dates.bin', 'rb') as f:
  packed_date = f.read(struct.calcsize('i3si'))

print(packer.unpack(packed_date))
```

Feladat1

- Készítsünk egy kórházi adatlap-kikereső alkalmazást, aminek
  - első parancssori paraméterként kell átadni az „adatbázis”-fájlt, amely egy bináris fájl
  - második paraméterként azt, hogy hányadik sort szeretnénk,
  - harmadik paramétertől kezdve pedig az oszlopokat, amelyekhez tartozó mezők értékét kell visszaadni az alkalmazásnak.
  - Oszlopok: 1. TAJ, 2. kor, 3. testhőmérséklet, 4. házas, 5. neme, 6. kezelés kezdetének ideje
- A sorok közvetlenül egymást követik és '9sif?c10s' formátummal lettek beírva a fájlba
- Az adatbázisfájl letölthető: **database.bin**
- Példa futtatás: python structfeladat.py database.bin 3 5 1 4
  - (4. sor 6., 2. és 5. oszlopa jelenik meg a kimeneten)

Szorgalmi feladat

- Az előző alkalmazást fejlesszük tovább/alakítsuk át, hogy
  - továbbra is parancssori paraméterként kelljen megadni az adatbázisfájlt, de a második paraméterként azt, hogy olvasni („R”) vagy írni szeretnénk („W”),
  - ha olvasás móddal lett indítva, akkor a kívánt sor és annak oszlopa már standard inputról jöjjön (most elég csak egy oszlopot megadni),
  - ha írás móddal lett indítva, akkor a felhasználó tudjon megadni új sort standard inputról való olvasással és legyen hozzáfűzve az adatbázisfájl végéhez.

PYTHON SOCKET PROGRAMOZÁS I.

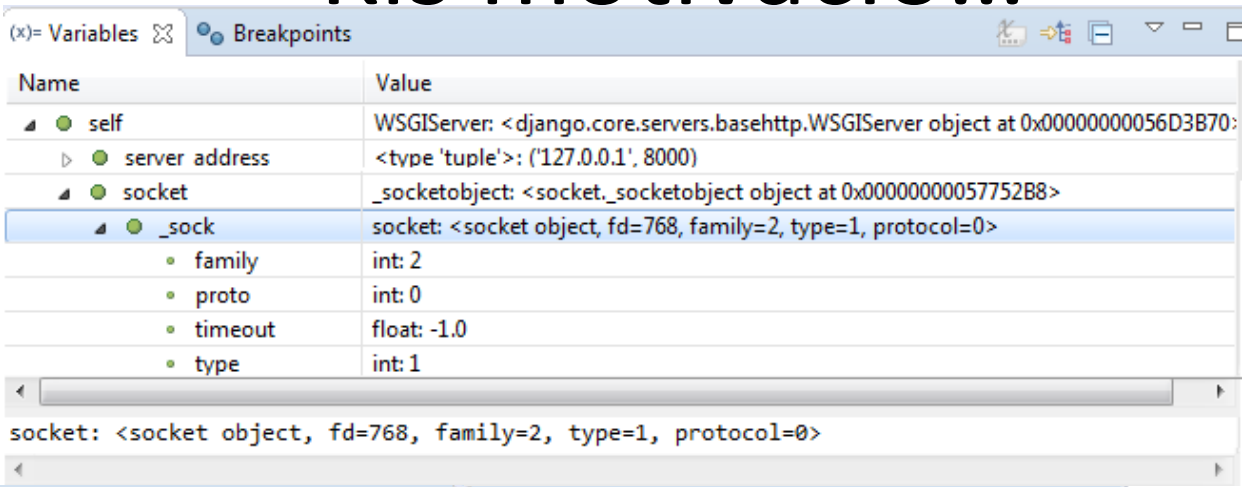
Kis motiváció…

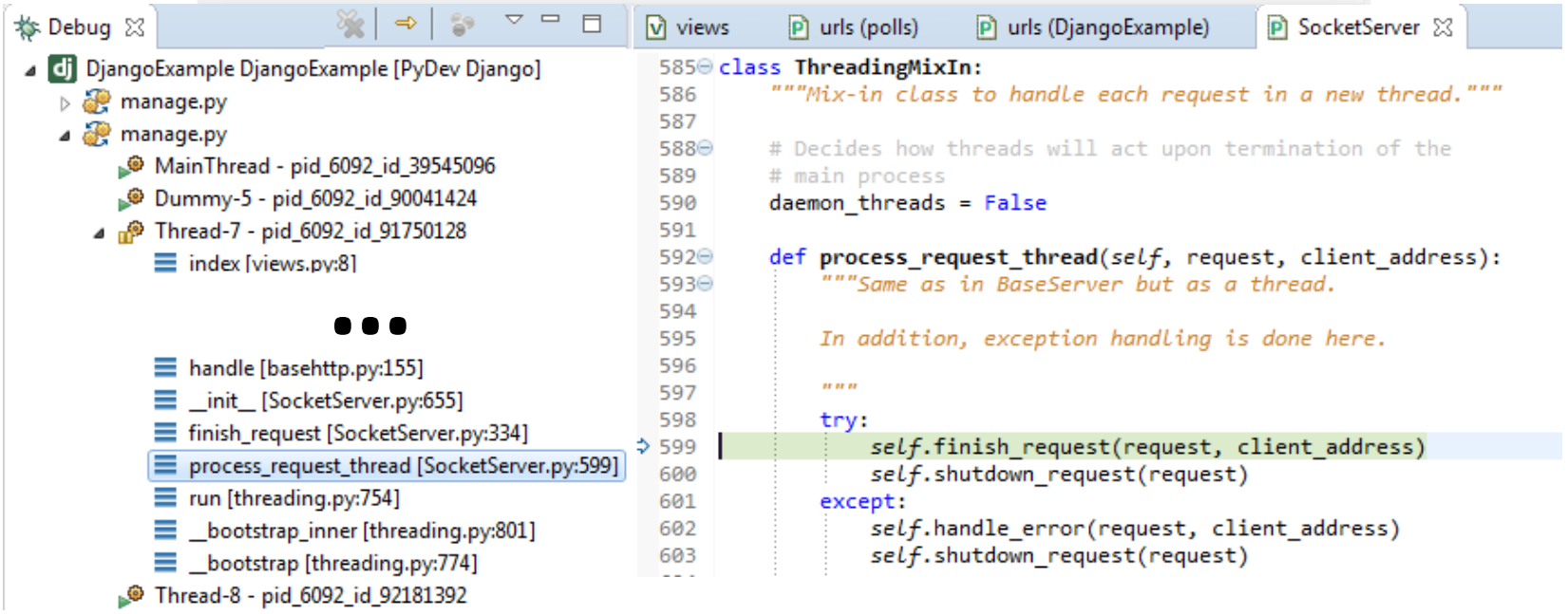
Kis motiváció…

views | urls (polls)

```
1  # -*- coding: utf-8 -*-
2  from __future__ import unicode_literals
3
4  from django.shortcuts import render
5
6  from django.http import HttpResponse
7
8  def index(request):
9      return HttpResponse("Hello ELTE!")
```

Kis motiváció…





Socket programozás

- Mi az a *socket*?
- A socket-ekre **kommunikációs végpontokként** gondolhatunk különböző programok közti adatcsere esetén (amelyek lehetnek ugyanazon a gépen, vagy különböző gépeken). Ezeket különböző operációs rendszerek támogatják, mint például: Unix/Linux, Windows, Mac stb.
- Egy program a saját socket-én keresztül ír és olvas egy másik program socket-jére/socket-jéről ⇒ az **adatátvitelt a socket-ek intézik** egymás között.
- Ezeket jól lehet használni a **kliens-szerver** modellnél, ahol a kliens valamilyen szolgáltatást igényel, amelyet a szerver szolgál ki. (Például a kliens egy weboldalt igényelhet HTTP protokollon keresztül egy webszervertől.)

A végpontok azonosítása, címzése hálózaton I.

- Két socket kommunikációjához ⇒ ismerni kell egymás azonosítóit
- Két részből állhat: IP címből és port számból
  - Az **IP címek** (IPv4 protokoll) 4 bájtos egész számok (pl. 157.181.152.1), amelyek (egyértelműen) azonosítják a gépeket, amelyek csatlakoznak az Internethez
  - Nehéz lenne megjegyezni ezeket ⇒ nevekre hivatkozunk helyettük (pl. www.elte.hu), amelyekhez tartozó IP címekre történő leképezéseket egy domain name server tud elvégezni
  - Spec. jelentéssel bír a localhost (127.0.0.1)

A végpontok azonosítása, címzése hálózaton II.

- Két socket kommunikációjához ⇒ ismerni kell egymás azonosítóit
- Két részből állhat: IP címből és port számból
  - Bizonyos protokollokhoz tartoznak fix portszámok, konstansok (szállítási protokollok)!
  - A **port számok** 2 bájtos egész számok, amelyek a gépen belül futó alkalmazásoknak az azonosításában segítenek
  - A portok közül vannak, amelyek foglaltak (pl. ftp a 21-es port, ssh a 22-es, http 80-as...)
  - A 0-s portnak spec. jelentése van ⇒ az OS keres egy szabad portot

Pyton socket, host név feloldás

- Socket csomag használata

```
import socket
```

- gethostname()

```
hostname = socket.gethostname()
```

- gethostbyname()

```
hostip = socket.gethostbyname('www.example.org')
```

- gethostbyname_ex()

```
hostname, aliases, addresses = socket.gethostbyname_ex(host)
```

- gethostbyaddr()

```
hostname, aliases, addrs = socket.gethostbyaddr('157.181.161.79')
```

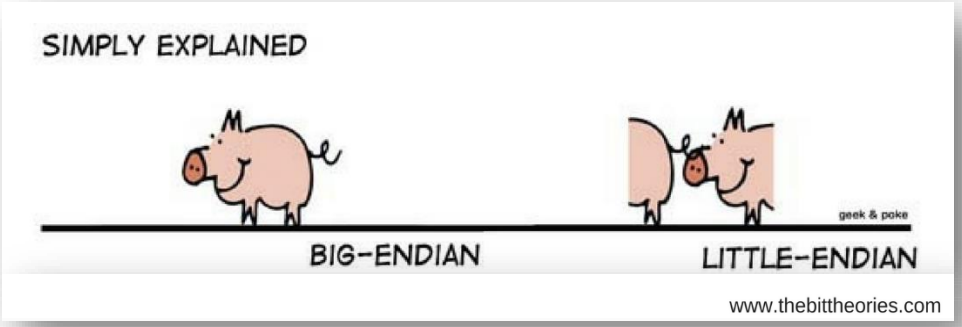
Port számok

- Bizonyos protokollokhoz tartoznak fix portszámok, konstansok (szállítási protokollok)!
- getservbyport()

```
socket.getservbyport(22)
```

- Írassuk ki a 1..100-ig a portokat és a hozzájuk tartozó protokollokat!

Bájtsorrendek



00000000 00000000 00000100 00000001

Address	Big-Endian representation of 1025	Little-Endian representation of 1025
00	00000000	00000001
01	00000000	00000100
02	00000100	00000000
03	00000001	00000000

- A hálózati bájt sorrend big-endian (a magasabb helyi értéket tartalmazó bájt van elől)
- Hoszt esetén bármi lehet: big- vagy little-endian
- Konverzió a sorrendek között:
  - 16 és 32 bites pozitív számok kódolása
  - socket.htons(), socket.htonl() – host to network short / long
  - socket.ntohs(), socket.ntohl() – network to host short / long

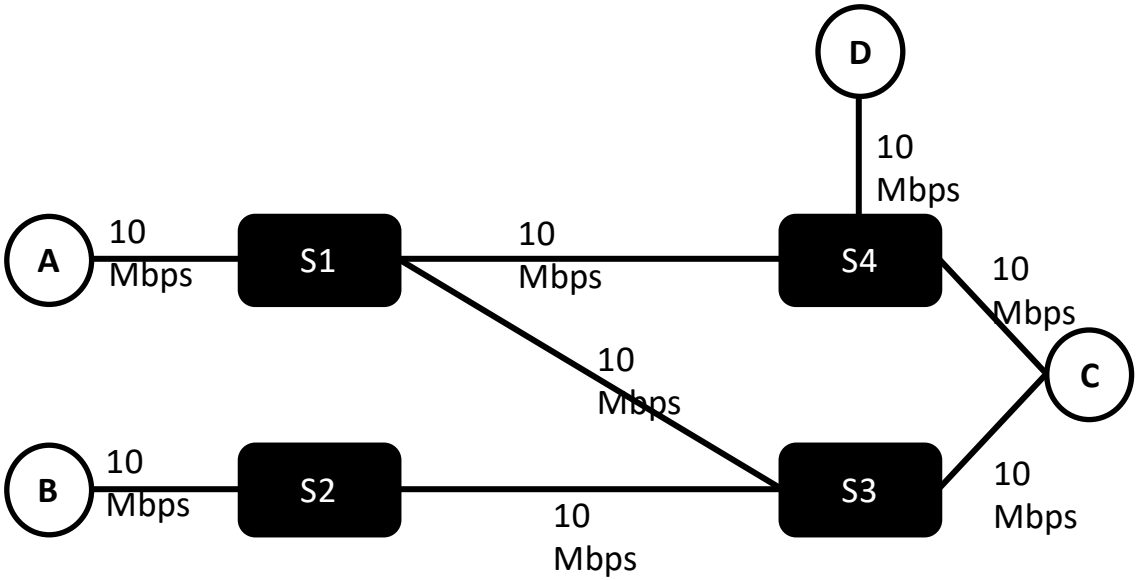
Fontos információ

- A házi feladatokat Python3-ban kell megírni, mert a TMS rendszeren keresztül elérhető tesztkörnyezet is abban van!

Áramkörkapcsolt hálózatok

HÁZI FELADAT I.

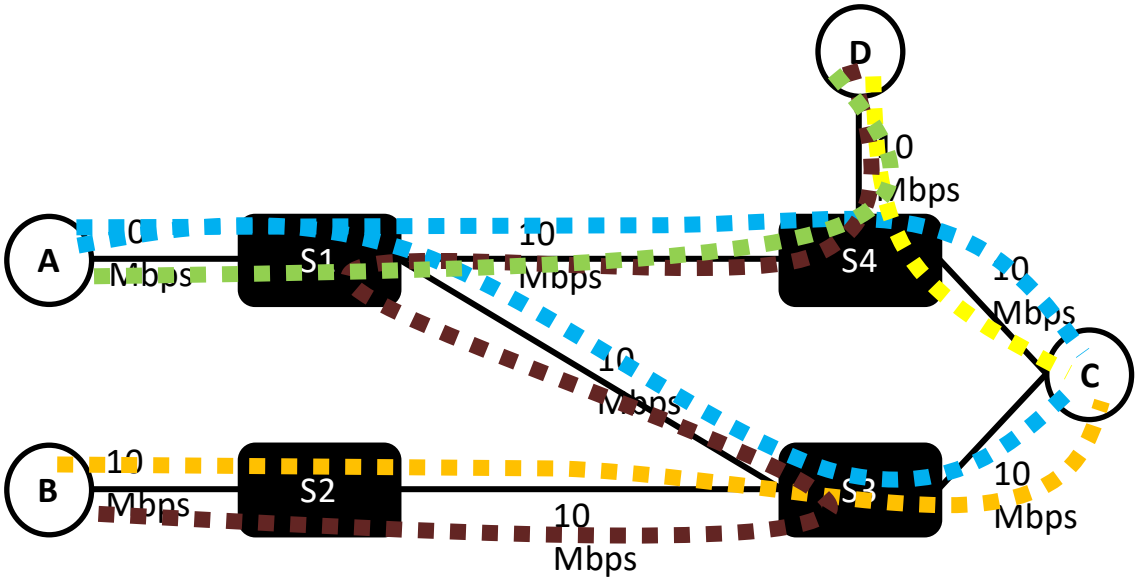
Topológia – cs1.json

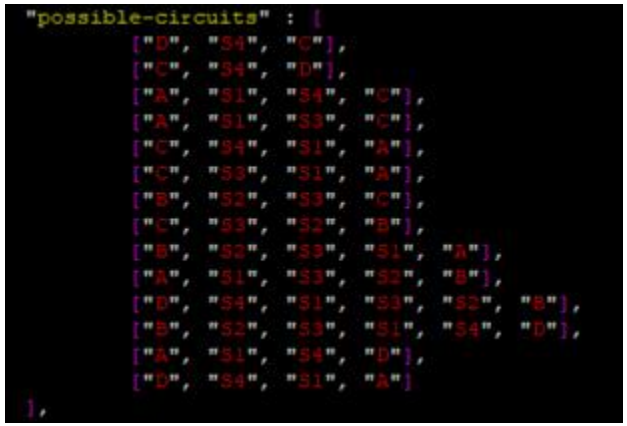


Irányítatlan legyen a gráf!!!

```
{
        "end-points": [ "A", "B", "C", "D" ],
        "switches": [ "S1", "S2", "S3", "S4" ],
        "links" : [
                {
                        "points" : [ "A", "S1" ],
                        "capacity" : 10.0
                },
                {
                        "points" : [ "B", "S2" ],
                        "capacity" : 10.0
                },
                {
                        "points" : [ "D", "S4" ],
                        "capacity" : 10.0
                },
                {
                        "points" : [ "S1", "S4" ],
                        "capacity" : 10.0
                },
                {
                        "points" : [ "S1", "S3" ],
                        "capacity" : 10.0
                },
                {
                        "points" : [ "S2", "S3" ],
                        "capacity" : 10.0
                },
                {
                        "points" : [ "S4", "C" ],
                        "capacity" : 10.0
                },
```

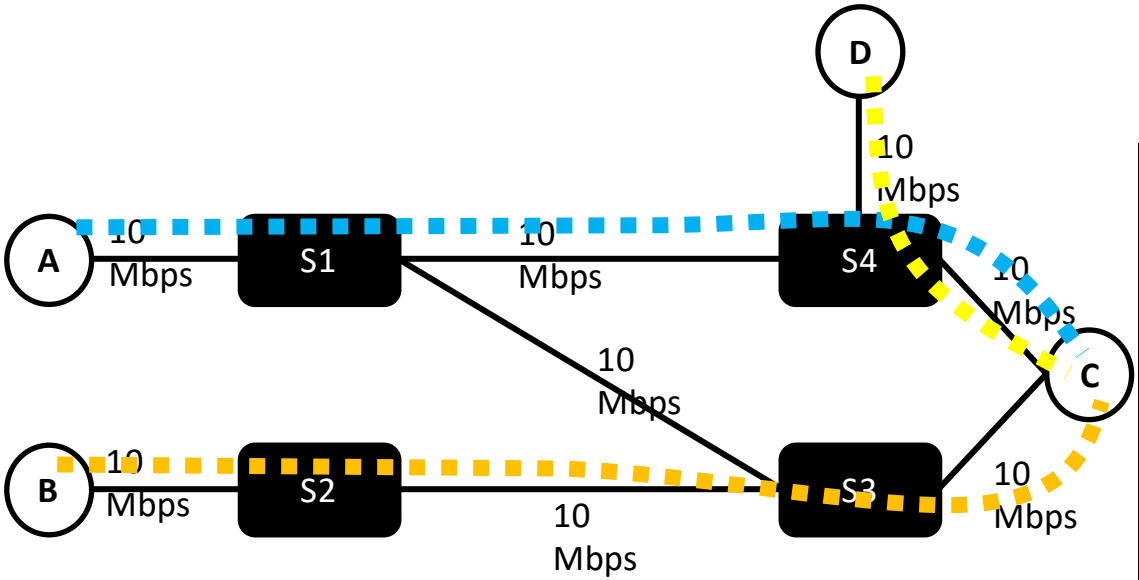
Lehetséges áramkörök – cs1.json





Irányítatlan legyen a gráf!!!

Igények – cs1.json



Irányítatlan legyen a gráf!!!

```
"simulation" : {
        "duration" : 11,
        "demands" : [
                {
                        "start-time" : 1,
                        "end-time" : 5,
                        "end-points" : ["A", "C"],
                        "demand" : 10.0
                },
                {
                        "start-time" : 2,
                        "end-time" : 10,
                        "end-points" : ["B", "C"],
                        "demand" : 10.0
                },
                {
                        "start-time" : 6,
                        "end-time" : 10,
                        "end-points" : ["D", "C"],
                        "demand" : 10.0
                }
        ]
}
```

Feladat

Adott a cs1.json, ami tartalmazza egy irányítatlan gráf leírását. A gráf végpont (end-points) és switch (switches) csomópontokat tartalmaz. Az élek (links) kapacitással rendelkeznek (valós szám). Tegyük fel, hogy egy áramkörkapcsolt hálózatban vagyunk és valamilyen RRP-szerű erőforrás foglaló protokollt használunk. Feltesszük, hogy csak a linkek megosztandó és szűk erőforrások. A json tartalmazza a kialakítható lehetséges útvonalakat (possible-cicuits), továbbá a rendszerbe beérkező, két végpontot összekötő áramkörigényeket kezdő és vég időponttal. A szimuláció a t=1 időpillanatban kezdődik és t=duration időpillanatban ér véget.

Készíts programot, ami leszimulálja az erőforrások lefoglalását és felszabadítását a JSON fájlban megadott topológia, kapacitások és igények alapján!

Script paraméterezése: python3 client.py <cs1.json>

A program kimenete:

Minden igény lefoglalását és felszabadítását írassuk ki a stdout-ra. Foglalás esetén jelezzük, hogy sikeres vagy sikertelen volt-e. Megj.: sikertelen esetben az igénnyel más teendőnk nincs, azt eldobhatjuk. Tehát:

<esemény sorszám. <esemény név>: <node1><-><node2> st:<szimuálciós idő> [- <sikeres/sikertelen>]

Pl.:

1. igény foglalás: A<->C st:1 – sikeres
2. igény foglalás: B<->C st:2 – sikeres
3. igény felszabadítás: A<->C st:5
4. igény foglalás: D<->C st:6 – sikeres
5. igény foglalás: A<->C st:7 – sikertelen

…

Leadás: A program leadása a TMS rendszeren .zip formátumban, amiben egy client.py szerepeljen!

Határidő: TMS rendszerben

VÉGE
KÖSZÖNÖM A FIGYELMET!