Szétválasztási hálózatok kiindulási struktúrájának automatikus generálása

Dokumentáció

Bóta Dániel Tamás, N9MERI

Feladatkitűzés

A félévben a feladatom egy olyan szoftver fejlesztése volt, aminek segítségével tetszőleges paraméterekkel automatikusan le lehet generálni egy szétválasztási hálózatot, pontosabban egy ilyen hálózat kiindulási struktúráját.

Ilyen hálózatokat sok területen használnak, a különböző munkafolyamatok, termelés optimalizálására. Meglévő adatokból, legyen az például az alapanyagok felépítése, összetevői, a mennyiségük, a munkagépek fajtája, kapacitása, működtetési költsége, a munkaórák száma, befektetési költségek stb., fel tudunk építeni ilyen hálózatokat, amiknek segítségével optimalizálni tudjuk a munkafolyamatokat, költségeket.

Ilyen hálózatok felépítésénél az első lépések közé tartozik a kiindulási-, vagy szuperstruktúra felépítése.

Egy ilyen hálózatnak több fontos eleme van. Áll a bemeneti, vagy feed streamekből, amik minden esetben több komponensűek – ezek általában valamilyen nyersanyagok vagy alapanyagok, az egyes komponensek pedig az adott anyag összetevőit képviselik, de ugyanakkor akármilyen tulajdonságot tekinthetnénk komponensnek, az adott feladattól függően.

A hálózat részei közé tartoznak még a separatorok, vagy szétválasztók, a dividerek, azaz elosztók és a mixerek, másnéven a keverők.

A szétválasztók fontos tulajdonsága, hogy egy bemenettel és két kimenettel rendelkeznek, ahol a kimenetek minden esetben a bemenet részhalmazai és a két kimenet különböző komponensekből áll össze. Más szóval, a bemenetet két részre választja és a kimenetek között nincs átfedés.

Az elosztók egy bemenettel és tetszőleges számú kimenettel rendelkeznek, ahol a kimenetek komponensei megegyeznek a bemenet komponenseivel. Ezzel ellentétben pedig a keverők több bemenettel és egy kimenettel rendelkeznek, ahol a bemenetek minden esetben a kimenet részhalmazai. A keverők kimenetei általában megegyeznek a gyártani kívánt termék komponenseivel.

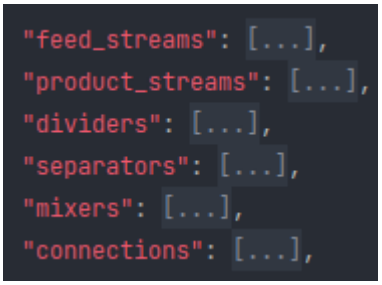
Az én feladatom volt, hogy tetszőleges komponensekkel rendelkező bemenetre (feed stream) és tetszőleges komponensekkel rendelkező kimenetre (product stream) egy olyan szoftvert fejlesszek, ami a megadott adatok alapján legenerálja a kiindulási struktúrát, annak felépítését egy adatszerkezetben eltárolja és a hálózat rajzát webes környezetben megjeleníti.

Felhasznált technológiák

A szoftver Python programozási nyelven készült, a PyCharm fejlesztői környezetet használva. Kizárólag a Python beépített könyvtárait használtam, nincs szükség külső modulokra.

Tervezés

Első lépésként meg kellett tervezni az adatstruktúrát, amiben a hálózat felépítését el lehet tárolni. Egy JSON struktúrát gondoltam a legmegfelelőbbnek, mivel ezt Python dictionary-vel és a Python beépített függvényeivel nagyon könnyen fel lehet építeni. A fájl tartalmazza a hálózat minden elemét: feed streamek, product streamek, separatorok, dividerek, mixerek, valamint a köztük lévő összeköttetések. Minden elem tartalmazza a bemeneti és kimeneti komponenseit (a divideréknél és mixereknél ezek megegyeznek, így nincsenek külön tárolva), valamint a separatorok esetében hogy „hol vág", azaz hol választja szét a bemeneti komponenseket.



Második lépés volt a struktúrát felépítő algoritmus implementálása, ami legenerálja az előbb említett hálózatot és azt eltárolja a megfelelő felépítésű JSON fájlba.

Ezután szükség volt egy olyan formátumra, amivel meg lehet rajzolni a hálózatot az felépített JSON alapján. Az SVG-re esett a választás, mivel viszonylag könnyen felépíthető és egyszerűen megjeleníthető webes felületen.

Megvalósítás

A program minden adatot egy osztályban tárol, aminek felépítése többé-kevésbé megegyezik az imént leírt JSON-el. Minden adatot listákban tárol el. Minden elemnek szintúgy van saját osztálya, ami minden esetben tartalmazza a komponenseit valamint egy UUID-t, ami alapján az összeköttetéseket azonosítani lehet. Ezeket a kapcsolatokat a Connection osztály segítségével tároljuk, ami eltárolja az adott 2 elem ID-jét. A „layers" és „max_dividers_in_single_layer" változók majd a későbbiekben, a hálózat megrajzolásakor lesznek szükségesek.

```
self.feed_streams = []
self.product_streams = []
self.dividers = []
self.separators = []
self.mixers = []
self.connections = []
self.layers = 0
self.max_dividers_in_single_layer = 0
```

A hálózat felépítése a következő algoritmussal történik:

1. Minden feed streamhez létrehozunk egy dividert. A dividerek komponensei megegyeznek az adott feed stream komponenseivel.
2. Kiválasztunk egy még nem vizsgált dividert.
3. Minden lehetséges elválasztáshoz létrehozunk egy separatort. Ha például a divider komponensei A, B, és C, akkor létre kell hozni két separatort: A-BC és AB-C.
4. Minden újonnan generált separatorhoz létrehozunk két dividert, a megfelelő komponensekkel. Például az A-BC separatorhoz létrehozandó két divider komponensei A és BC. Ezzel a 2. lépésben kiválasztott dividert vizsgáltnak tekintjük.

5. Ha van még vizsgálatlan divider, visszatérünk a 2. lépéshez. Ha nincsen, az azt jelenti, hogy a hálózat fel van építve.

Ez az algoritmus programkódban így néz ki:

```
while True:
    count = 0
    new_separators = []
    # iterate through each divider
    for div in ss.dividers:
        if div not in ignored_divs:  # create separators only if they don't already exist
            if len(div.comps) > 1:
                for i in range(1, len(div.comps)):
                    new_sep = Separator(div.comps, i, layers)
                    new_con = Connection(div.id, new_sep.id)
                    ss.separators.append(new_sep)
                    new_separators.append(new_sep)
                    ss.connections.append(new_con)
                ignored_divs.append(div)
        else:  # count ignored dividers
            count += 1

    if count == len(ignored_divs):
        break

    no_of_new_dividers = 0
    # iterate through each separator
    for sep in new_separators:  # create dividers only if they don't already exist
        # all separators need 2 dividers
        new_div1 = OpUnit(sep.output[0], layers)
        new_div2 = OpUnit(sep.output[1], layers)
        ss.dividers.append(new_div1)
        ss.dividers.append(new_div2)
        ss.connections.append(Connection(sep.id, new_div1.id))
        ss.connections.append(Connection(sep.id, new_div2.id))
        no_of_new_dividers += 2
    if no_of_new_dividers > max_dividers_in_single_layer:
        max_dividers_in_single_layer = no_of_new_dividers
    layers += 1
```

Végig iterál a dividereken, és a még nem vizsgáltak esetében végrehajtja a fent említett algoritmust. A ciklus akkor áll le, ha nem talál olyan dividert, amit még meg kéne vizsgálnia. A ciklus minden iterációval eltárolja az adott „layert” – ez tulajdonképpen csak azt jelenti, hogy minden elemnél tudjuk, hogy hányadik iterációval jött létre, ami majd a rajzolást könnyíti meg. Ugyanilyen a max_dividers_in_single_layer is – rajzolás során az elemek elhelyezését segíti.

A program ezután a kész hálózatot lementi egy JSON fájlba, azaz kiolvassa az adatokat a Structure osztályból, átírja JSON-nek megfelelő formátumba és lementi a fájlt a projekt könyvtárába.

A program rajzoló része a generált Structure osztállyal és a JSON fájllal dolgozik.

A hálózat elemei alapvetően egyszerű alakzatokkal ábrázolhatók. Minden separator egy téglalap, minden divider és mixer pedig háromszög. Ezek az alakzatok SVG formátumban, XML-hez hasonló tagekkel könnyen leírhatók:

```
<rect x="100" y="35" width="100" height="35" style="fill:rgb(255,255,255);stroke-width:1;stroke:rgb(0,0,0)"></rect>
<polygon points="220,100 250,135 190,135" style="fill:rgb(255,255,255);stroke-width:1;stroke:rgb(0,0,0)"></polygon>
```

Hogy a programban könnyebben lehessen ezekkel dolgozni, minden egy-egy osztály reprezentál, ami tartalmazza az adott elem koordinátáit a síkon, adott esetben a szélességét, hosszúságát, a formázható stílusát, ID-jét, valamint hogy melyik layeren helyezkedik el. Így az SVG struktúra felépíthető ezen osztályok és konstruktoraik segítségével. A fájlba való íráshoz minden osztály tartalmaz egy függvényt, ami visszaadja az SVG-nek megfelelő formátumot:

```
self.x = x
self.y = y
self.width = 100
self.height = 35
self.style = "fill:rgb(
self.id = id
self.layer = layer
```

```
def add_to_svg(self):
    return f"<rect x='{self.x}' y='{self.y}' width='{self.width}' height='{self.height}' style='{self.style}' />"
```

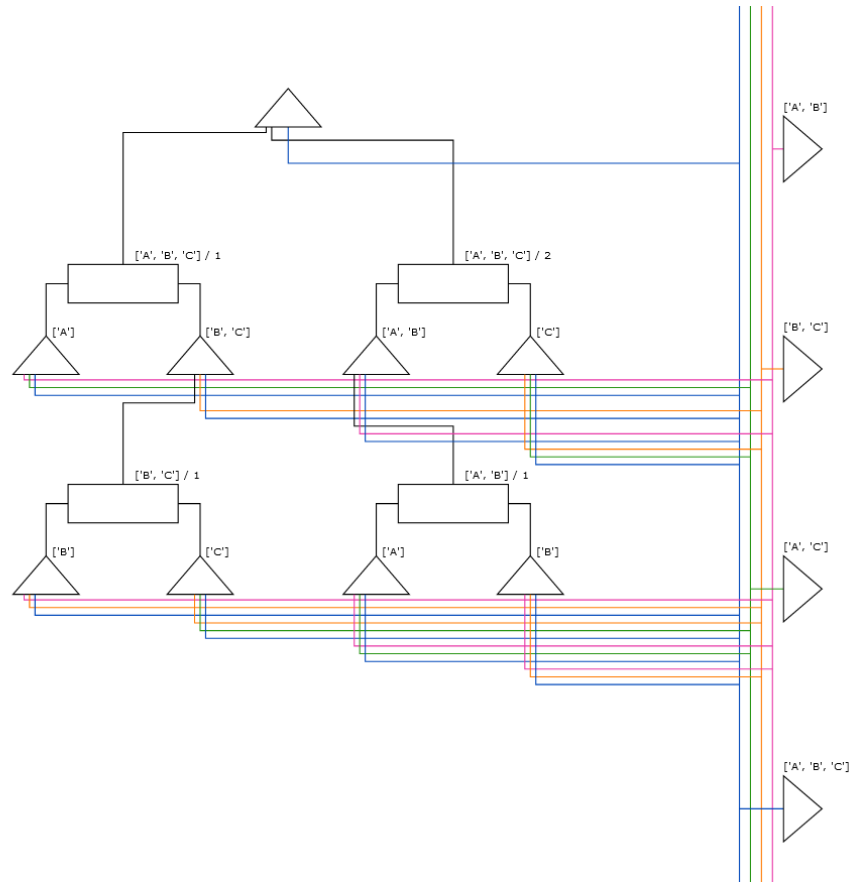
Mivel egyesével nehéz lenne az elemeket a síkon elhelyezni úgy, hogy azok átláthatóak legyenek, az algoritmus első lépésként csoportosítja őket: minden separatorhoz megkeresi az ahhoz tartozó 2 dividert és ezeket egybe kezeli. Mivel egy adott csoporton belül az elemek relatív pozíciója mindig megegyezik, vagyis a csoportok egyformák, ezeket különösebb számolás nélkül le lehet generálni és a későbbiekben elég a csoportok helyét meghatározni a síkon. Az összetartozó elemeket egyszerűen meg lehet keresni az ID-jük alapján.

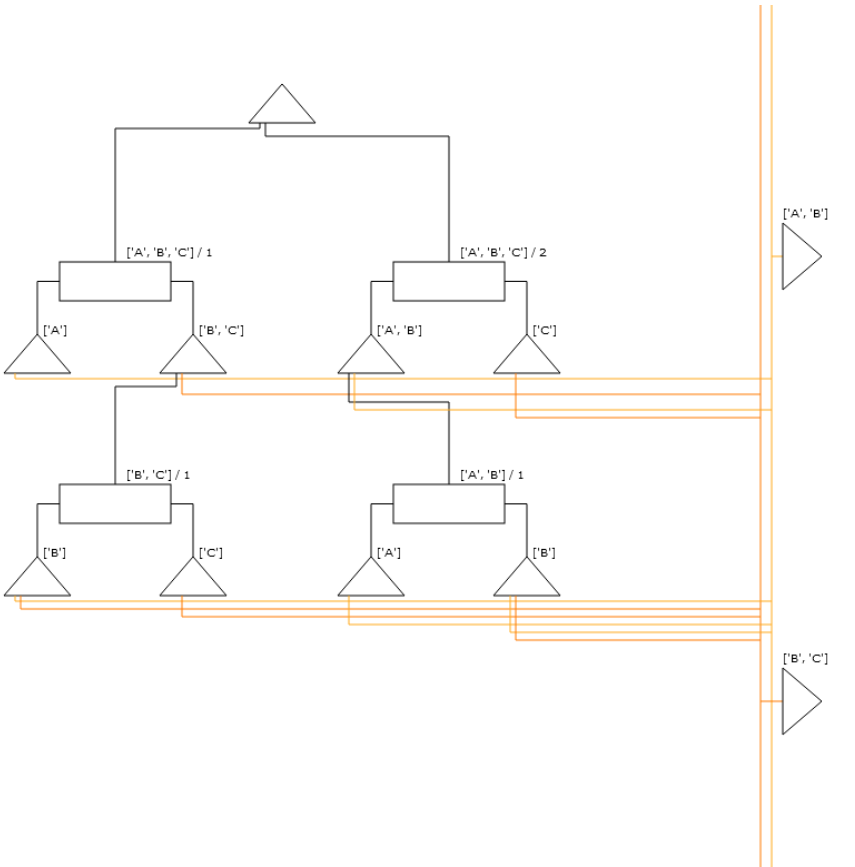
A korábban eltárolt layers változó alapján meg lehet határozni a vászon méretét és mivel tudjuk, hogy melyik separator melyik layeren helyezkedik el (így az adott csoportok elhelyezkedését is ismerjük), ki tudjuk számolni a csoportok koordinátáit a síkon.

```
i = 0
# place each group on the plane
for j in groups_per_layer:
    for k in range(j):
        svg_groups[i].x = k * 300
        svg_groups[i].y = (svg_groups[i].sep.layer - 1) * 200
        svg.groups.append(svg_groups[i])
        i += 1
```

A feed streameket a csoportok fölé helyezzük el, a mixereket pedig a csoportok mellé. Ennek oka a dividerekből jövő bypassek. Bypassról beszélünk, amikor a divider komponenshalmaza részhalmaza valamelyik mixer komponenseinek. Ilyenkor közvetlen összekötjük a dividert és a mixert. Mivel ez rengeteg összeköttetést jelent, főleg nagyobb hálózatokon, a bypasseket színkódolva rendezetten jobbra vezetjük el, hogy a hálózat minél átláthatóbb legyen. Hogy az összeköttetések ne csússzanak egybe, a program számontart egy 5-7 egységnyi eltoltást minden új kapcsolat megrajzolásakor, amiket lenulláz minden új layeren.

Mindennek végeredményeként az alábbihoz hasonló hálózatok jöhetnek létre:





Tesztelés

A program tesztelve lett 1-3 feed streamre, 1-4 mixerre, maximum 5 komponensre (itt már nagyon nagy hálózatokról beszélünk) és minden tesztre helyes eredményt adott, átlátható hálózattal.

Hátralévő feladatok

- A csoportok elhelyezésén javítani: ez az y tengelyre vonatkozik leginkább, sok bypass esetén előfordulhat, hogy egyes összeköttetések és separatorok között átfedés van
- Grafikus felületet létrehozni a user inputnak: egyenlőre csak parancssorban lehet megadni a feed- és product streameket

Források, felhasznált dokumentumok:

- Heckl István - Szétválasztási hálózatok szintézise: Különböző tulajdonságokon alapuló szétválasztó módszerek egyidejű alkalmazása (https://konyvtar.uni-pannon.hu/doktori/2007/Heckl_Istvan_dissertation.pdf)

- Network Synthesis and Optimization Course 2019 (https://www.youtube.com/watch?v=-wzGOzjqXpE&list=PLRA5Vy7Ua4tevOIFIzEUZeAsA3aNMfwv-)