4. modul – Az igraph és a pylab modulok használata

Összetett hálózatok vizsgálata

Horváth Árpád <horvath.arpad@amk.uni-obuda.hu>

Óbudai Egyetem
Alba Regia Műszaki Kar (AMK)
Székesfehérvár

2016. november 1.

Vázlat

Gráfok létrehozása az igraph használatával

Gráfok kirajzoltatása, a gráf, az élek és a csúcsok jellemzői

A GraphML formátum, szomszédok, részgráfok, gyakorlás

Komponensek és átmérő meghatározása az igraph használatával

A modul célja

A hallgatók ismerjék meg a igraph és pylab modulok használatát olyan szinten, amelyek szükségesek a hálózatok interaktív vizsgálatához illetve a hálózatok vizsgálatát végző programok írásához.

Megjegyzések a modulhoz

Az egyes leckékben gyakran példa mutatja be az igraph illetve a pylab lehetőségeit. Egy leckén belül gyakran építünk a korábban a leckében létrehozott objektumokra, ezért az egyes leckéket egyszerre érdemes feldolgozni anélkül, hogy az ipython3 parancsértelmezőt bezárnánk. Külön leckék esetén mindig figyelmeztetünk rá, ha újra létre kell hozni valamilyen objektumot.

Vázlat

Lecke: Gráfok létrehozása az igraph használatával

Tervezett időtartam 90 perc.

Követelmények: A hallgató

- létre tud hozni él nélküli irányított és irányítatlan gráfokat akárhány csúccsal,
- létre tud hozni irányított és irányítatlan gráfokat éllistából,
- hozzá tud adni, illetve törölni tud éleket és csúcsokat egyesével és többet egyszerre,
- meg tudja állapítani, hogy összefüggő-e a gráf,
- meg tudja állapítani, hogy egyszerű-e a gráf, van-e benne többszörös illetve hurokél,
- meg tudja állapítani a hálózat csúcsainak és éleinek a számát,
- képes kiszámoltatni az értelmezővel a hálózat átlagfokszámát.

Feladat

Mielőtt elolvasná, hogyan tud létrehozni élek nélküli hálózatokat, hogyan adhatja meg, hogy irányított legyen-e, hogyan adhat hozzá éleket, próbálja meg kitalálni ezeket az ipython3 parancsértelmező segítségével! Importálja az igraph modult, és kérjen segítséget a igraph.Graph osztályról!

A dokumentációs karakterlánc (docstring) értelmezése

Ha kérdőjelet ír a igraph.Graph osztály neve után ipython3-ban, akkor megjelenik többek között az osztály docstring-je, amiben példák is találhatóak, és a konstruktor docstring-je, amelyben a lehetséges paramétereket látja felsorolva. A konstruktor docstring-je tartalmazza a paramétereket:

```
__init__(n=None, edges=None, directed=None,
  graph_attrs=None,
  vertex_attrs=None, edge_attrs=None)
```

Ezalatt megtalálhatóak az egyes paraméterek leírásai. Láthatjuk, hogy megadhatjuk létrehozáskor a csúcsok számát (n), létrehozhatunk éleket éllistából (edges), megadhatjuk, hogy irányított legyen-e a gráf (directed), illetve megadhatjuk a gráf, a csúcsok illetve élek jellemzőit.

Feladat

A továbbiakban leírásra kerülnek az alábbi feladatok megoldásai, de először próbálja meg önállóan kitalálni, csak a docstring segítségével.

Hozzon létre irányítatlan illetve irányított gráfot 6 csúccsal élek nélkül!

Megoldás

Változatok él nélküli 6 csúcsú gráfok létrehozása.
Irányítatlan:

```
net = igraph.Graph(6)
net = igraph.Graph(6, directed=False)
net = igraph.Graph(6, directed=0)
```

Irányított:

```
dirnet = igraph.Graph(6, directed=True)
dirnet = igraph.Graph(6, directed=1)
```

Az alsóknál kihasználtuk, hogy True/False helyett bármi állhat, ami arra értékel ki a bool „függvény”:
`bool(0)` értéke `False`, `bool(1)` értéke `True`.

A summary függvény

Az előbbi feladat esetén nem tudtuk ellenőrizni, hogy azt kaptuk-e, amit szerettünk volna. Az ellenőrzés egyik lehetőségét az igraph.summary függvény biztosítja. Ez paraméterként a gráfot várja, és kiír egy összefoglalót a hálózatról. Az előző példa dirnet gráfja esetén a következőt írja ki:

```
igraph.summary(dirnet)
IGRAPH D--- 6 0 --
```

Ebben a 6 jelenti a csúcsok számát, a 0 az élek számát. A D betű jelzi, hogy irányított hálózatról van szó (egyébként U betű szerepel).

Gráfok létrehozása éllistából

A gráfokat létrehozhatjuk éllistából is. A konstruktor docstringjéből láthatjuk, hogy a második paraméter az edges, tehát azt vagy hely szerinti paraméterként használhatjuk, vagy kulcsszavas paraméterként. De ha egyetlen paraméter van, akkor ha az lista, akkor azt éllistaként értelmezi.

```
net = igraph.Graph(8, [(0,1), (1,2), (3,4)])
net = igraph.Graph(edges=[(0,1), (1,2), (3,4)])
net = igraph.Graph([(0,1), (1,2), (3,4)])
```

Az utóbbi két esetben a legnagyobb előforduló csúcsindex alapján találja ki, hogy hány csúcsa legyen a gráfnak. Mivel a példában ez 4, azért 5 csúccsal hozza létre a gráfot, hiszen a legelső csúcs azonosítója 0. Az első változatban az éllista miatt létrehozott 0, 1, 2, 3 és 4 indexű csúcsokat kiegészíti további izolált csúcsokkal, hogy a csúcsok száma az első paraméterrel legyen egyenlő.

A get_edgelist metódus

A létrehozott gráfok éllistáját létrehozhatjuk a get_edgelist metódussal, ami hasznos lehet, ha ellenőrizni szeretnénk a továbbiakban a gráf változásait. A másik ellenőrzési lehetőséggel, a gráfok kirajzoltatásával, csak a következő leckében ismerkedünk meg.

```
net.get_edgelist()  # eredménye [(0, 1), (1, 2), (3, 4)]
```

Csúcsok hozzáadása és törlése

Csúcsok hozzáadásakor csak a csúcsok számát kell megadni. Az előző 8 csúcsú hálózatot 10 eleművé bővíthetjük az alábbi paranccsal:

```
net.add_vertices(2)
```

Ekkor a hálózat újabb két izolált csúccsal fog bővülni, a 8-as és 9-es indexűekkel. A csúcsok törlésekor a delete_vertices függvénynek a csúcsok indexeit kell megadni listaként, vagy egyetlen csúcsindexet. Például:

```
net.delete_edges([7,8])
```

Figyelni kell arra, hogy ilyenkor az indexelés folytonos marad, a 9-es indexű csúcs lesz a 7-es indexűvé.

Éllista megváltoztatása

Az igraph adatszerkezete olyan, hogy nagyon gyorsan meg tudja határozni egy csúcs fokszámát és szomszédait, de az új élek hozzáadása lassú. Lényegében egyszerre 100 élt hozzáadni nagyjából ugyanannyi idő, mint egyet. Általában is igaz, hogy az igraph-ban lassú bármely olyan művelet, amely az éllista megváltozásával jár: az élek törlése, és a nem izolált csúcsok törlése is, és ezeket célszerű tömegesen végezni, ha lehet: egyszerre több élt, csúcsot törölni.
Az élek hozzáadása többnyire az add_edges metódussal történik, bár létezik egy él hozzáadására alkalmas add_edge metódus is. A törlésre csak a többes számú alak, a delete_edges létezik.

Élek hozzáadása és törlése

Az add_edges és a delete_edges metódusok, legegyszerűbb alakjukban, éllistát várnak paraméterként, azaz csúcsindex-párok listáját.
Az alábbi példában létrehozunk egy gráfot, és azt több lépésben megváltoztatjuk. Érdemes lépésenként előre kitalálni, hogy mi történik az egyes sorokban, hogyan néz ki az egyes lépések után a gráf. Ehhez a get_edgelist metódust és a summary függvényt is segítségül hívhatjuk.

Feladat

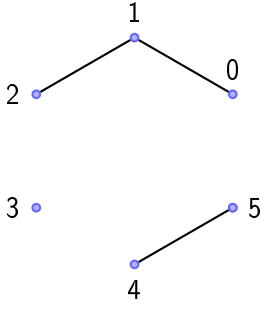
Milyen lesz az egyes lépések után a gráf? Rajzolja le!

```
net = igraph.Graph([(0,1), (1,2), (4,5)])
net.delete_vertices([3])
net.add_edges([(1,4), (0,4), (2,3)])
net.delete_edges([(0,1), (2,1)])
net.delete_vertices([3])
net.add_edges([(1,2)])
net.delete_edges()
```

Segítség a megoldáshoz

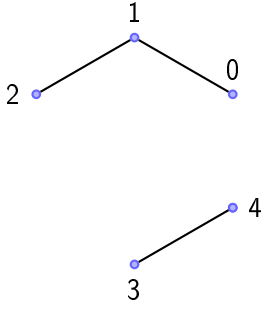
A delete_vertices sorok után az indexelés megváltozik. Mint említettük az indexelés mindig folytonos marad. Esetünkben mindkét törlésnél a 3-asnál nagyobb indexűek indexe csökken eggyel.
A delete_edges paraméter nélküli hatását nem tárgyaltuk, de a get_edgelist és a summary megmutatja, hogy az összes él törlődik, és persze a csúcsok megmaradnak.

Megoldás



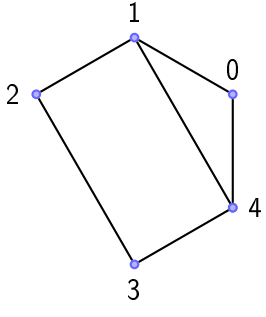

```
net = igraph.Graph([(0,1), (1,2), (4,5)])
```

Megoldás



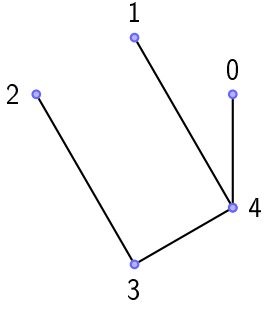

```
net = igraph.Graph([(0,1), (1,2), (4,5)])
net.delete_vertices([3])
```

Megoldás



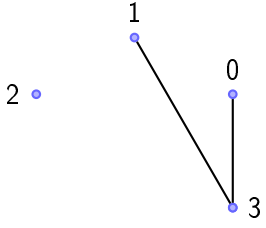

```
net = igraph.Graph([(0,1), (1,2), (4,5)])
net.delete_vertices([3])
net.add_edges([(1,4), (0,4), (2,3)])
```

Megoldás



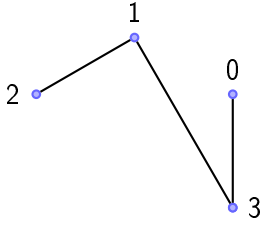

```
net = igraph.Graph([(0,1), (1,2), (4,5)])
net.delete_vertices([3])
net.add_edges([(1,4), (0,4), (2,3)])
net.delete_edges([(0,1), (2,1)])
```

Megoldás



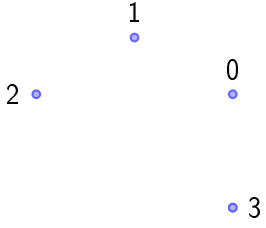

```
net = igraph.Graph([(0,1), (1,2), (4,5)])
net.delete_vertices([3])
net.add_edges([(1,4), (0,4), (2,3)])
net.delete_edges([(0,1), (2,1)])
net.delete_vertices([3])
```

Megoldás




```
net = igraph.Graph([(0,1), (1,2), (4,5)])
net.delete_vertices([3])
net.add_edges([(1,4), (0,4), (2,3)])
net.delete_edges([(0,1), (2,1)])
net.delete_vertices([3])
net.add_edges([(1,2)])
```

Megoldás




```
net = igraph.Graph([(0,1), (1,2), (4,5)])
net.delete_vertices([3])
net.add_edges([(1,4), (0,4), (2,3)])
net.delete_edges([(0,1), (2,1)])
net.delete_vertices([3])
net.add_edges([(1,2)])
net.delete_edges()
```

Globális tulajdonságok vizsgálata

A Graph osztály rendelkezik számos paraméterrel, amellyel megvizsgálhatóak, hogy egy hálózat például összefüggő vagy irányított-e. Ezeknek a metódusoknak a neve általában is_ karakterhármassal kezdődik.

Feladat

Milyen metódusai vannak a Graph osztálynak, amelyek is_ kezdetűek?

Segítség

Az ipython3 a `Tab` (tabulátor, általában a `CapsLk` felett) gomb egyszeri megnyomásával kiegészíti a metódus (objektum) nevét, amíg egyértelmű, kétszeri megnyomással pedig kilistázza a lehetséges folytatásokat.

Megoldás

A net.is_ beírása után a Tab-gomb kétszeri megnyomásával többek között a következő metódusokat kapjuk (csak a metódusok neve net. nélkül):

is_directed is_simple is_multiple is_loop
is_connected is_named is_weighted is_bipartite

Feladat

Az alábbi metódusok közül melyikkel vizsgálhatjuk meg, hogy egy hálózatot irányítottként hoztunk-e létre, illetve, hogy jelen állapotában összefüggő-e a hálózat?

is_directed is_simple is_multiple is_loop
is_connected is_named is_weighted is_bipartite

Megoldás

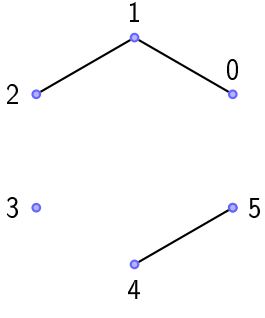
Az is_directed True értéket ad, ha irányított a gráf, különben False értéket. Az is_connected az összefüggőséget vizsgálja hasonló módon.

Feladat

A következő lépéseket már vizsgáltuk egyszer. Az akkor elkészített ábrák alapján határozzuk meg, és ellenőrizzük, hogy melyik lépés után ad az is_connected metódus True illetve False értéket? (Az újra-begépelés helyett használjuk ki, hogy pl. a net.d begépelése után a felfelé gomb csak a net.d kezdetű sorok között keres.)

```
net = igraph.Graph([(0,1), (1,2), (4,5)])
net.delete_vertices([3])
net.add_edges([(1,4), (0,4), (2,3)])
net.delete_edges([(0,1), (2,1)])
net.delete_vertices([3])
net.add_edges([(1,2)])
net.delete_edges()
```

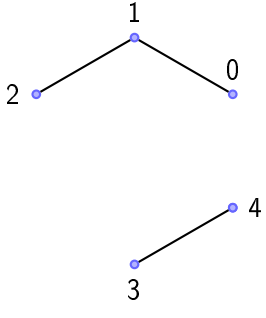
Megoldás



net.is_connected() == False

```
net = igraph.Graph([(0,1), (1,2), (4,5)])
```

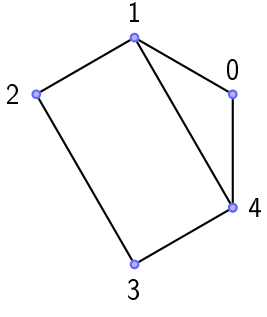
Megoldás



net.is_connected() == False

```
net = igraph.Graph([(0,1), (1,2), (4,5)])
net.delete_vertices([3])
```

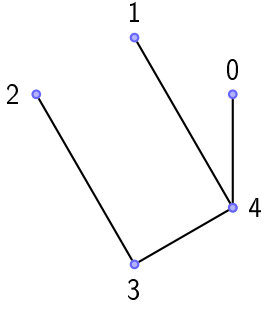
Megoldás



net.is_connected() == True

```
net = igraph.Graph([(0,1), (1,2), (4,5)])
net.delete_vertices([3])
net.add_edges([(1,4), (0,4), (2,3)])
```

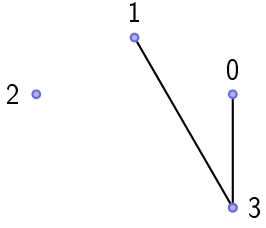
Megoldás



net.is_connected() == True

```
net = igraph.Graph([(0,1), (1,2), (4,5)])
net.delete_vertices([3])
net.add_edges([(1,4), (0,4), (2,3)])
net.delete_edges([(0,1), (2,1)])
```

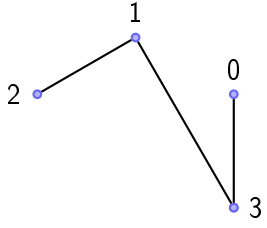
Megoldás



net.is_connected() == False

```
net = igraph.Graph([(0,1), (1,2), (4,5)])
net.delete_vertices([3])
net.add_edges([(1,4), (0,4), (2,3)])
net.delete_edges([(0,1), (2,1)])
net.delete_vertices([3])
```

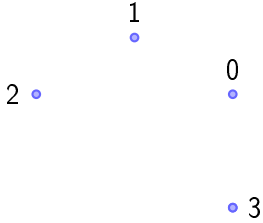
Megoldás



net.is_connected() == True

```
net = igraph.Graph([(0,1), (1,2), (4,5)])
net.delete_vertices([3])
net.add_edges([(1,4), (0,4), (2,3)])
net.delete_edges([(0,1), (2,1)])
net.delete_vertices([3])
net.add_edges([(1,2)])
```

Megoldás



net.is_connected() == False

```
net = igraph.Graph([(0,1), (1,2), (4,5)])
net.delete_vertices([3])
net.add_edges([(1,4), (0,4), (2,3)])
net.delete_edges([(0,1), (2,1)])
net.delete_vertices([3])
net.add_edges([(1,2)])
net.delete_edges()
```

Többszörös és hurokélek

Az igraph nem csak egyszerű gráfokat képes kezelni, hanem olyanokat is, amelyben van többszörös él, vagy hurokél, vagy akár mindkettő.
Az élek törlésekor, ha többszörös élt adunk meg, csak az egyiket törli. Ha élek létrehozásakor már létező élt adunk meg, akkor az él többszöröződik. A gráf létrehozásakor egy él annyiszoros lesz, ahányszor az éllistában szerepel.

Feladat

Vajon melyik metódusokkal tudjuk megállapítani egy gráfról, hogy egyszerű-e, hogy van-e benne többszörös él, illetve hurokél?

Megoldás

A korábban szerepeltek az is_ kezdetű metódusok.

Metódus	Mi vizsgálható vele?
is_simple	Egyszerű-e a gráf?
is_multiple	Van-e többszörös él benne?
is_loop	Van-e hurokél benne?

A második kettő (ha argumentum nélkül hívjuk meg) logikai értékek listáját adja vissza!

Vizsgálódások az any függvénnyel

Az is_multiple és is_loop metódusok minden egyes élre megmondják, hogy többszörös illetve hurokél-e, tehát logikai értékekből álló listát adnak. Az is_multiple függvény minden élnél, amely legalább másodjára szerepel, True értéket ad vissza. Ha először szerepel, akkor is False értéket ad, ha később még szerepel ugyanolyan végpontokkal rendelkező él. Ez a furcsa viselkedés egyszerűvé teszi majd a többszörös élek olyan törlését, hogy minden összekötött csúcspár között pontosan egy él maradjon.
Az Python nyelvben az any függvény vizsgálja, hogy egy listában van-e igaz érték.

```
net = igraph.Graph([(0,1), (0,1), (0,1)])
net.is_multiple()      # [False, True, True]
any(net.is_multiple()) # True, van többszörös él
any(net.is_loop())     # False, nincs hurokél
```

Feladat

Ha nem lenne is_simple metódus, mit írhatnék a következő feltételvizsgálat helyett?

net.is_simple()

Megoldás

Egy net nevű igraph.Graph osztálybeli objektum esetén az alábbi két sor egyenértékű (bár az alsó több időt vesz igénybe):

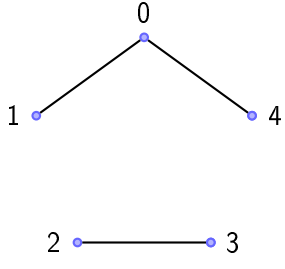
```
net.is_simple()
any(net.is_multiple()) or any(net.is_loop())
```

Feladat

Rajzolja le az egyes lépések után a gráfot! Minden lépésnél állapítsa meg, hogy a gráf összefüggő-e, tartalmaz-e többszörös élt illetve hurokélt, valamint egyszerű-e! A megállapításokat a tanult metódusok segítségével is ellenőrizze!

```
net = igraph.Graph([(0,1), (0,4), (3,2)])
net.add_edges([(2,2), (3,4)])
net.add_edge(2,3)
net.delete_vertices(2)
```

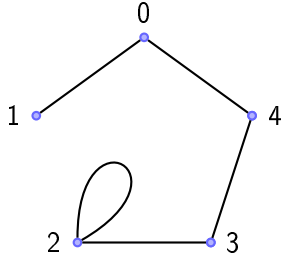
Megoldás



nem összefüggő,
egyszerű (nincs többszörös és hurokél)

```
net = igraph.Graph([(0,1), (0,4), (3,2)])
```

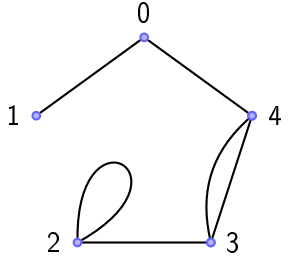
Megoldás



összefüggő,
nem egyszerű, nincs többszörös él de van hurokél

```
net = igraph.Graph([(0,1), (0,4), (3,2)])
net.add_edges([(2,2), (3,4)])
```

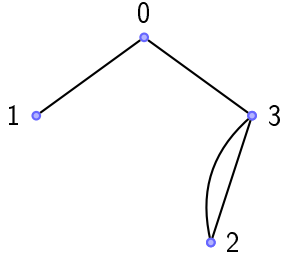
Megoldás



összefüggő,
nem egyszerű, van többszörös él és hurokél

```
net = igraph.Graph([(0,1), (0,4), (3,2)])
net.add_edges([(2,2), (3,4)])
net.add_edge(4,3)
```

Megoldás



összefüggő,
nem egyszerű, van többszörös él de nincs hurokél

```
net = igraph.Graph([(0,1), (0,4), (3,2)])
net.add_edges([(2,2), (3,4)])
net.add_edge(4,3)
net.delete_vertices(2)  # Két csúcs indexe változik!
```

A vcount és ecount metódusok

A summary metódus lehetőséget nyújt ugyan arra, hogy parancssorban megtudjuk egy hálózat vagy gráf csúcsainak és éleinek a számát, gyakran van viszont programokban szükség arra, hogy ezekkel az értékekkel számoljunk. Ilyenkor jön jól a igraph.Graph osztályból származó objektumok vcount és ecount metódusa. Képletekben a csúcsok számát gyakran kis vagy nagy N betűvel, az élekét pedig gyakran kis vagy nagy M betűvel jelölik. Mi a nagybetűs jelölésmódot használjuk, mivel a m jelölést másra tartjuk fenn.

```
net = igraph.Graph([(0,4), (4,5)])
N = net.vcount()    # N = 6
M = net.ecount()    # M = 2
```

A halmazok számossága

A halmazok számosságára gyakran a halmazok jele köré tett abszolútérték-jelekkel utalnak. Mivel a matematikában a csúcsok halmazát gyakran V betűvel jelölik, az élek halmazát pedig E-vel, ezért az N és M jelölések helyett gyakran használják a $|V|$ illetve $|E|$ jelöléseket is. A függőleges vonal viszont nem használható változónévben, ezért gyakori programokban az N és M jelölés.

Ismételjük át az átlagfokszámot, átlagos be- és kifokszámot, és a kiszámításukra vonatkozó képleteket. Az i. csúcs fokszámát k_i-vel jelöljük. Irányítottnál pl. a befokszámát $k_{be,i}$-vel. A csúcsok számát N-nel jelöljük, és a csúcsokat 0-tól indexeljük. Az átlagfokszám alatt a fokszámok számtani közepét értjük, amely képlettel:

.

Az átlagos befokszám alatt a befokszámok számtani közepét értjük, amely képlettel:

.

Ismételjük át az átlagfokszámot, átlagos be- és kifokszámot, és a kiszámításukra vonatkozó képleteket. Az i. csúcs fokszámát k_i-vel jelöljük. Irányítottnál pl. a befokszámát $k_{be,i}$-vel. A csúcsok számát N-nel jelöljük, és a csúcsokat 0-tól indexeljük. Az átlagfokszám alatt a fokszámok számtani közepét értjük, amely képlettel:

$$\langle k \rangle = \frac{\sum_{i=0}^{N-1} k_i}{N} = \frac{\text{fokszámok összege}}{\text{csúcsok száma}} \quad .$$

Az átlagos befokszám alatt a befokszámok számtani közepét értjük, amely képlettel:

.

Ismételjük át az átlagfokszámot, átlagos be- és kifokszámot, és a kiszámításukra vonatkozó képleteket. Az i. csúcs fokszámát k_i-vel jelöljük. Irányítottnál pl. a befokszámát $k_{be,i}$-vel. A csúcsok számát N-nel jelöljük, és a csúcsokat 0-tól indexeljük. Az átlagfokszám alatt a fokszámok számtani közepét értjük, amely képlettel:

$$\langle k \rangle = \frac{\sum_{i=0}^{N-1} k_i}{N} = \frac{\text{fokszámok összege}}{\text{csúcsok száma}} \quad .$$

Az átlagos befokszám alatt a befokszámok számtani közepét értjük, amely képlettel:

$$\langle k_{be} \rangle = \frac{\sum_{i=0}^{N-1} k_{be,i}}{N} = \frac{\text{befokszámok összege}}{\text{csúcsok száma}} \quad .$$

Feladat

Mi a kapcsolat az

1. élek száma (M),
2. a csúcsok száma (N) és
3. az átlagos fokszám ($\langle k \rangle$)

között?
Mi a helyzet, ha az átlagfokszám helyett a $\langle k_{be} \rangle$ átlagos befokszám vagy a $\langle k_{ki} \rangle$ átlagos kifokszám érdekel?

M Megoldás jön! Lapozás előtt próbálja megoldani a feladatot!

Kapcsolat a hálózatok alapvető tulajdonságai között

Az éleknek kép végpontja van, tehát minden egyes él kettővel növeli a fokszámok összefüggét.

Az átlagos fokszám:

Kapcsolat a hálózatok alapvető tulajdonságai között

Az éleknek kép végpontja van, tehát minden egyes él kettővel növeli a fokszámok összefüggét.

Az átlagos fokszám:

Kapcsolat a hálózatok alapvető tulajdonságai között

Az éleknek kép végpontja van, tehát minden egyes él kettővel növeli a fokszámok összefüggét.

Az átlagos fokszám: $\langle k \rangle = \frac{2M}{N}$

Befokszám esetén az átlagos fokszám:

Kapcsolat a hálózatok alapvető tulajdonságai között

Az éleknek kép végpontja van, tehát minden egyes él kettővel növeli a fokszámok összefüggét.

Az átlagos fokszám: $\langle k \rangle = \frac{2M}{N}$

Befokszám esetén az átlagos fokszám:

Kapcsolat a hálózatok alapvető tulajdonságai között

Az éleknek kép végpontja van, tehát minden egyes él kettővel növeli a fokszámok összefüggét.

Az átlagos fokszám: $\langle k \rangle = \frac{2M}{N}$

Befokszám esetén az átlagos fokszám:

$\langle k_{be} \rangle = \frac{M}{N}$

Kifokszám esetén szintén.

Feladat

Írjunk egy olyan programsort, amely egy net nevű hálózat esetén (az igraph.Graph egy objektuma) kiszámítja az átlagfokszámot!

Írjunk egy olyan programsort, amely irányított hálózat esetén kiszámítja az átlagos befoszámot!

M Megoldás jön! Lapozás előtt próbálja megoldani a feladatot!

Megoldás

A net hálózat átlagfokszáma illetve átlagos befokszáma:

```
k_avg = 2*net.ecount() / net.vcount()
k_in_avg = net.ecount() / net.vcount()
```

Összefoglalás – Hálózat létrehozása

Él nélküli

```
empty = igraph.Graph()
isolated = igraph.Graph(5)
net = igraph.Graph(5, directed=True)  # irányított
```

Éllistából

```
net = igraph.Graph([(2,3), (3,4)])
net2 = igraph.Graph(5, [(2,3), (3,4)])
net = igraph.Graph([(2,3), (3,4)], directed=True)
```

Összefoglalás – Hálózat módosítása

```
net.add_vertices(5)  # 5 csúcs hozzáadása
net.delete_vertices([2,5,7,8])
net.delete_vertices(5)   # 5-ös csúcs törlése
net.delete_edges([(2,3), (3,4)])
net.delete_edges([(2,3)])
net.add_edges([(2,1), (0,2)])
```

Összefoglalás – Hálózat vizsgálata

```
net.is_connected()
net.is_simple()
net.is_directed()
any(net.is_multiple())
any(net.is_loop())
N = net.vcount()
M = net.ecount()
```

Vázlat

Lecke: Gráfok kirajzoltatása, a gráf, az élek és a csúcsok jellemzői

Tervezett időtartam 90 perc.

Követelmények: A hallgató

- ki tud rajzoltatni gráfokat,
- el tud menteni gráfábrákat fájlokba,
- ismeri a PNG, az SVG és PDF formátumok előnyeit, hátrányait egymáshoz képest,
- fel tudja sorolni a csúcsok és élek kirajzolását befolyásoló fontosabb csúcs- és éljellemzőket (három csúcs- és két éljellemző),
- tudja használni a csúcs és éljellemzőknek megfelelő kulcsszavas argumentumokat,
- képes használni a layout kulcsszavas argumentumot,
- létre tud hozni és le tud kérdezni gráf, él és csúcsjellemzőket,
- ki tud válogatni adott jellemzőjű csúcsokat és éleket,
- listázni tudja az él-, a csúcs- és a gráfjellemzőket,
- értelmezni tudja a igraph.summary(net) függvény kimenetét,
- be tud tölteni hálózatokat tartalmazó fájlokat.

A python-igraph dokumentáció

pyigraph tutorial Az igraph Pythonos felületéhez van egy angol nyelvű oktató oldal, tutorial, amely a kurzusban használtaknál bővebb leírást tartalmaz az igraphról. A kurzusban a hatékony használathoz szükséges minimumot célozzuk meg, és közel sem említünk minden lehetőséget. Ezt a tutorialban találják. Szintén megtalálhatnak ott számos dolgot táblázatos összefoglaló formában.
Ha rákeresnek a python igraph tutorial szavakra a google-ben, akkor könnyen megtalálják, de a teljesség kedvéért mellékeljük az url-jét

`http://hal.elte.hu/~nepusz/development/igraph/tutorial/tutorial.html`

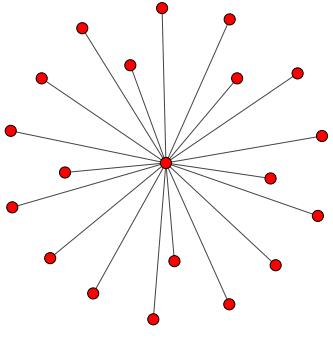
Bevezetés

Az igraph képes irányítatlan és irányított gráfok kirajzolására. A kirajzolás jellemzői finoman hangolhatóak kétféle módon: egyik esetben a kirajzoltatandó függvénynek adjuk meg a rajzolás paramétereit kulcsszavas paramétereit, vagy a gráf-objektumban tárolhatjuk azokat. Az utóbbihoz meg kell ismerkednünk a VertexSeq (csúcssorozat) és EdgeSeq (élsorozat) osztállyal, és azok jellemzőinek használatával. Erről fog szólni a lecke.

A plot függvény

Egy gráfot kirajzoltathatunk a plot paranccsal. Az alábbi egy úgynevezett csillag-gráfot hoz létre és rajzol ki. (ipython köv. oldalon)

```
import igraph
net = igraph.Graph([(0,i) for i in range(1,20)])
igraph.plot(net)
```



Tipp: plot parancs az ipythonban

Ha terminálból indítjuk az ipython3-at, akkor nem biztos, hogy az

```
igraph.plot(net)
```

sor hatására megjelenik az ábra. Ilyenkor fűzzük a show metódust a végére:

```
igraph.plot(net).show()
```

Mentés fájlba

Az előbbi esetben a plot függvényt egyetlen argumentummal, egy gráffal hívtuk meg. Ilyenkor a csúcsok színe és mérete, az élek vastagsága és számos más tudajdonság alapértelemzett érték, az ábrát valamelyik képmegjelenítő program jeleníti meg.
A második paraméterként megadhatunk egy fájlnevet. A kiterjesztés alapján a plot függvény meghatározza, milyen formátumba kell kimentenie. Ezek lehetnek PNG, PDF, SVG és PostScript. Mi az első hárommal foglalkozunk. Amennyiben a fájlnevek .png, .pdf illetve .svg kiterjesztésűek, a megfelelő formátumba íródik ki az ábra. A fájlnév előtt útvonal is lehet: "img/star.png" esetén az aktuális könyvtár img alkönyvtárába ment.

A PDF és SVG formátumok

A PDF és SVG fájlformátumok vektorgrafikus formátumok, a geometriai objektumokat nem pixelenként, hanem paraméterekkel tárolják. Például egy egyenest a végpontok koordinátáival. A vektorgrafikus formátumok tehát olyan ábrák tárolására hatékonyak, amelyek geometriai objektumokból állnak. A gráfok ábrázolásai pont ilyenek: egyenesek, körök, háromszögek (irányított gráf nyílhegyei) és betűk szerepelnek általában benne. Az SVG formátum a Web hivatalos formátuma, akár a HTML5-kód része is lehet SVG-ábra.
A PDF régebbi és kiforrottabb formátum az SVG-nél. A LaTeX dokumentumformázó nyelv, amivel ez a dokumentum is készült, képes PDF-ábrák használatára. Gyakran a webböngészők képesek (önállóan vagy plug-innel) megjeleníteni PDF-ábrákat, de nem az oldal részeként.

A PNG fájlformátum

A PNG fájlformátum úgynevezett pixelgrafikus formátum. A képeket sorokba és oszlopokba rendezett képpontonként, pixelenként tárolja: minden pixelnél a háromféle színösszetevő (piros-zöld-kék, angol rövidítéssel RGB) értékét. A kis fájlméret érdekében az így kapott adatokat még veszteségmentes módon tömöríti. A PNG-formátum szintén használható weboldalon belül.

A csúcssorozat és az élsorozat

Az igraph-ban minden egyes csúcsnak illetve minden egyes élnek egy indexe van. Mindkét esetben az indexelés folytonos (nullától darabszámnál eggyel kisebb számig), törléskor a megmaradóak indexei változhatnak. A többszörös élek minden egyes példányának külön indexe van. A net Graph-objektum esetén net.vs jelöli a gráf csúcssorozatát, a net.es az élsorozatát. Ezeket indexelhetjük a listák, tömbök indexelésének szabályai szerint. Mindegyik csúcsnak (élnek) egyesével, és az egész sorozatnak is adhatunk különböző jellemző értékeket. Például az alábbi sorok a 0-ás indexű csúcs "color" jellemzőjét "blue" értékre, a 0-ás indexű él "color" jellemzőjét "green" értékre állítja.

```
net.vs[0]["color"] = "blue"
net.es[0]["color"] = "green"
```

A kirajzolást befolyásoló jellemzők

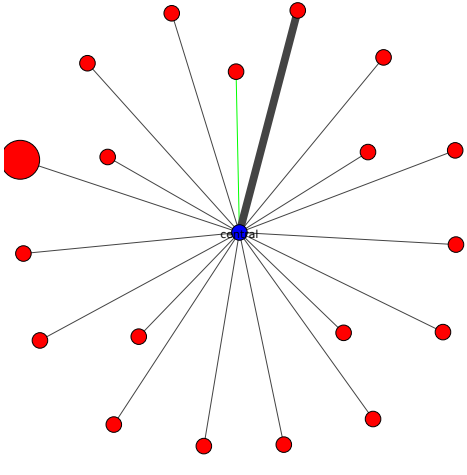
A fenti módosítás után, ha újra kirajzoltatjuk a gráfot, akkor a középső csúcs és az egyik él színe megváltozik. Az alábbi táblázatban láthatóak azok a rajzolást befolyásoló fontosabb jellemzők. A továbbiakat a Graph.__plot__ docstringjében vagy az pyigraph tutorial -ban találhatják meg.

vertex	edge
size	width
color	color
label	

Feladat

Állítsa be a net gráf 0-ás indexű csúcsának label jellemzőjét "central" értékre, az 1-es indexű élének width jellemzőjét 10-re, a 3-as indexű csúcsának size jellemzőjét 50-re! Ezután rajzoltassa ki a gráfot! Ha nem találja valamelyik változtatás nyomát az ábrán, akkor vizsgálja meg, nem írt-e el valamit! (A width és size jellemzőt számként, a többit karakterláncként kell megadni.)

Segítség



Ha jól csinálta, az itt láthatóhoz hasonló ábrát kapott.
(Elképzelhető, hogy az itt láthatóhoz hasonlóan a nagy kör egy része kívül kerül az ábrán. Ezt majd a margó növelésével el lehet kerülni.)

Megoldás

```
net.vs[0]["label"] = "central"
net.vs[0]["size"] = 50
net.es[1]["width"] = 10
```

Vertex és Edge objektum

Ha a parancsértelmezőben beírjuk egy konkrét csúcsot vagy élt a megfelelő index-szel, akkor megláthatjuk a jellemzőit.

```
net.vs[0]
igraph.Vertex(<igraph.Graph object at 0xb4ccdac>, 0,
  {'color': 'blue', 'label': 'central', 'size': None})
net.es[0]
igraph.Edge(<igraph.Graph object at 0xb4ccdac>, 0,
  {'color': 'green', 'width': None})
```

A jellemzők

Látható, hogyha egy jellemzőt valamelyik csúcsnál (élnél) megadunk, akkor az a másik csúcsnál (élnél) is megjelenik, de az értéke None lesz. Az igraph minden jellemzőre egy lista-szerű objektumot (tömböt) hoz létre. Ezeket a tömböket az igraph C-ben írt része kezeli, csúcsok élek törlésekor C-program rendezi a megfelelő elemek törlését is.
A teljes listát kiírathatjuk illetve módosíthatjuk, ha a csúcs (él) indexét elhagyjuk.

```
net.vs["color"]
["blue", None, None ...]
net.es["width"]
[None, 10, None ...]
```

Jellemzőlista módosítása

Módosításkor az értékadás jobboldalán is listának lehet lista (vagy más sorozattípus). A lista lehet rövidebb is, mint a csúcsok (élek) száma, ilyenkor az értékek periodikusan ismétlődnek.

```
net.vs["color"] = ["red", "blue"]
```

Ekkor az érték felváltva "red" és "blue" lesz, ahogy arról a kiiratásával meggyőződhetünk.
Minden egyes jellemzőváltoztatás után rajzoltassa ki a gráfot is!
A csúcsokat cimkézhetjük az indexével a következő módon:

```
net.vs["label"] = range(net.vcount())
```

A label értékei lehetnek számok és karakterláncok is.

Egyforma jellemzőérték beállítása

Ha az összes csúcsnak (élnek) egy bizonyos jellemző esetén azonos értéket szeretnénk adni, akkor a jobb oldalra egyetlen számot, vagy karakterláncot írjunk.

```
net.vs["color"] = "green"
net.es["width"] = 3
```

Ezután az összes csúcsot zölddel és az összes élt azonos (3 pixel) vastagsággal fogja kirajzoltatni.

Egyéb jellemzők

A csúcsoknak és éleknek tetszőleges nevű jellemzőt beállíthatunk, de a kulcs (a jellemző neve) mindig karakterlánc legyen. Bonyodalmakhoz vezethet például az alábbi (pedig szótárnak lehetne kulcsa az (1,2) tuple):

```
net.vs[(1,2)] = 11
```

Feladat

Hozzunk létre egy s nevű (szociális) hálózatot, amelyben

- 10 csúcs van,
- saját magán kívül minden csúcs kapcsolódik a 9-es indexűhöz,
- ezen kívül a 3-ashoz kapcsolódik a 2-es és 4-es,
- minden páros indexű csúcs rózsaszín (pink), a páratlan indexűek kékek,
- a csúcsoknak van egy gender (nem) nevű jellemzőjük, ennek az értéke párosaknál "f" (nő), páratlanoknál "m" (férfi),
- a csúcsok mérete 20 (pixel).

Mentse el a hálózat ábráját social.png néven.

Egy lehetséges megoldás

```
edge_list = [(i,9) for i in range(9)] + [(2,3), (3,4)]
```

Egy lehetséges megoldás

```
edge_list = [(i,9) for i in range(9)] + [(2,3), (3,4)]
s = igraph.Graph(edge_list)
```

Egy lehetséges megoldás

```
edge_list = [(i,9) for i in range(9)] + [(2,3), (3,4)]
s = igraph.Graph(edge_list)
s.vs["color"] = ["pink", "blue"]
```

Egy lehetséges megoldás

```
edge_list = [(i,9) for i in range(9)] + [(2,3), (3,4)]
s = igraph.Graph(edge_list)
s.vs["color"] = ["pink", "blue"]
s.vs["gender"] = ["f", "m"]
```

Egy lehetséges megoldás

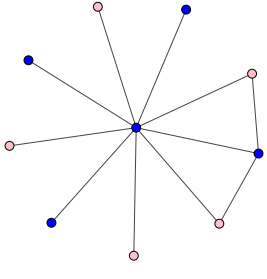
```
edge_list = [(i,9) for i in range(9)] + [(2,3), (3,4)]
s = igraph.Graph(edge_list)
s.vs["color"] = ["pink", "blue"]
s.vs["gender"] = ["f", "m"]
s.vs["size"] = 20
```

Egy lehetséges megoldás

```
edge_list = [(i,9) for i in range(9)] + [(2,3), (3,4)]
s = igraph.Graph(edge_list)
s.vs["color"] = ["pink", "blue"]
s.vs["gender"] = ["f", "m"]
s.vs["size"] = 20
igraph.plot(s, "social.png")
```

Egy másik lehetséges kezdés, és a kapott ábra

```
edge_list = [(i,9) for i in range(9)]
edge_list.extend([(2,3), (3,4)])
s = igraph.Graph(edge_list)
```



Színezés tulajdonság szerint

Gyakran előfordul, hogy a csúcsnak például a neme adott, és utólag szeretnénk azalapján például színnel elkülöníteni a pontokat. Ez egy szótár és egy listaértelmezés használatával megoldható ahogy az alábbi példán látható.

```
colors = {"m": "yellow", "f": "red"}
s.vs["color"] = [ color[g] for g in s.vs["gender"] ]
```

Feladat

Az előző példa mintájára alakítsa át az *s* szociális hálózatot úgy, hogy a férfiakat jelölő körök (gender=m) eltérő méretűek legyenek, mint a nőké!
Ajánlott feladat: úgy alakítsa át, hogy a férfiakat háromszög jelölje! (Segítség a Graph.__plot__ docstringjében.)

Megoldás

A feladat első részének megoldása:

```
sizes = {"f": 20, "m": 15}
s.vs["size"] = [sizes[g] for g in s.vs["gender"]]
```

vagy tömörebben:

```
s.vs["size"] = [{"f": 20, "m": 15}[g]
                for g in s.vs["gender"]]
```

Az ajánlott rész megoldása:

```
shapes = {"f": None, "m": "triangle"}
s.vs["shape"] = [shapes[g] for g in s.vs["gender"]]
```

Jellemzők megadása kulcsszavas argumetumokkal

A rajzolás során használt jellemzőket megadhatjuk, illetve felülírhatjuk kulcsszavas argumentumokkal. A csúcsjellemzők kulcsszavát úgy kapjuk, hogy a jellemző neve elé vertex_ szót fűzzük. Az éljellemzőknél természetesen edge_ kell a név elé.
Az alábbi példában minden csúcsot sötétsárgával fog kirajzoltatni, és az élek vastagságát egy számsorból kapjuk.

```
igraph.plot(s, vertex_color="yellow3", edge_width=range(1,12))
```

A hálózat jellemzői eközben nem változnak, tehát újra csak az s paraméterrel meghívva a régi ábrát kapjuk vissza.

Feladat

Rajzoltassa ki az s hálózatot úgy, hogy a csúcsok mérete legyen egy számtani sorozat, az élek színei pedig pirosak és kékek legyenek!

Egy lehetséges megoldás

```
igraph.plot(s, vertex_size=range(5, 5+s.vcount()),
    edge_color=["red", "blue"])
```

A csúcsok elrendezése

Láthattuk, hogy általában a plot függvény a csúcsokat mindig máshová helyezi. Az igraph-nak több algoritmusa van a csúcsok elhelyezésére. Vannak közülük, amelyek minden futáskor ugyanoda helyezik a csúcsokat (circle), de a legtöbb olyan, hogy próbálják jól elhelyezni a csúcsokat, de a végeredmény futásról futásra eltérhet.
pyigraph tutorial Az egyes algoritmusokat az igraph Python-os tutorialjában láthatjuk felsorolva a *Layouts and plotting* fejezetben.
A rövid név ismeretében például egy lépésben kirajzoltathatjuk a gráfot a layout kulcsszavas argumentummal, a circle kulcsszóval egy körön helyezkednek el a csúcsaink sorrendben, egyenletesen elosztva:

```
igraph.plot(s, layout="circle")
```

Elrendezések mentése és felhasználása

Vannak olyan algoritmusok, amely az összekapcsolt éleket próbálja egymáshoz közel elhelyezni, mintha erő vonzaná őket egymáshoz: egyik példa az igraph-ból a Kamada-Kawai algoritmus, rövid nevén "kk". Ezt közvetlenül is megadhatnánk a plot függvényben, de gyakran hasznos tárolnunk az elrendezést. Többek között a Graph-objektum layout metódusával hozhatunk létre egy elrendezést, és a tárolt elrendezés alapján mindig úgyanott lesznek a csúcspontjaink, ha a layout kulcsszavas argumentumnak a mentett értékket adjuk meg. A kapott kordinátákat ki is listázhatjuk.

```
lo = s.layout("kk")
igraph.plot(s, layout=lo)
list(lo)
```

Elrendezések létrehozása

Az elrendezéseket ki is listázhatjuk:

```
list(lo)
```

Sőt létre is hozhatunk elrendezéseket: annyi párnak kell benne szerepelnie, ahány csúcsunk van.

```
lo = [(0,0), (0,1), (0,2), (0,3), (0,4), (1,0), (1,1), (1,2), (1,3), (1,4)]
# vagy elegánsabban
lo = [(x,y) for x in (0,1) for y in range(5)]
igraph.plot(s, layout=lo)
```

További fontos jellemzők

Eddig láttunk olyan jellemzőket, amelyek a kirajzolást befolyásolták (color), olyanokat, amelyeknek csak a számunkra volt jelentősége (gender). Vannak olyan jellemzők, amelyek egyes algoritmusok lefutását befolyásolják, vagy a csúcsok megkeresését teszik könnyebbé. Az éljellemzők közül ilyen a weight jellemző amellyel súlyozott gráfot hozhatunk létre, és a type és name csúcsjellemzők. Például megbetűzhetjük a csúcsokat:

```
s.vs["name"] = list("abcdefghij")
```

Ezek egyike sem befolyásolja a kirajzolást, de megjeleníthetjük az ábrán bármelyiket. Például kiírathatjuk a neveket az alább jelzett módok egyikével:

```
s.vs["label"] = s.vs["name"]
igraph.plot(s, vertex_label=s.vs["name"])
```

Feladat

Súlyozza a gráfok éleit valamilyen számokkal, és ezeket a súlyokat a kirajzolásnál is jelenítse meg szemléletes módon!

Egy lehetséges megoldás

```
s.es["weight"] = [1,3,2,4,2,6,7,3,5,2,7]
igraph.plot(s, edge_width=[1+2*w  for w in s.es["weight"]])
```

Feladat

Kérjen egy összegzést az s hálózatról az előző leckében említett summary függvénnyel!

Megoldás

```
igraph.summary(s)
```

A summary értelmezése

Jelen eseben a következőhöz hasonló kimenetet adhat:

IGRAPH UNW- 10 11 --
+ attr: color (v), gender (v), label (v), name (v), size (v),
color(e), weight (e)

Itt az UNW- sorozat karakterei egyesével a következőt jelentik: irányítatlan (U, irányítottnál D lenne), van neve a csúcsoknak (N, name), súlyozott a gráf (W, weight) és nincs típusa a csúcsoknak (-, T lenne, ha lenne type). Hiányzó jellemző esetén az utóbbi háromnál - szerepel.
A hálózatnak adott jellemzőket is látjuk felsorolva, zárójelben látszik, hogy melyik csúcsjellemző (v) és melyik élé (e).

Gráfjellemzők

Nem csak az éleknek és csúcsoknak, hanem a gráfnak is adhatunk jellemzőket:

```
s["name"] = "social network"
s["date"] = "30-04-2014"
```

Feladat

Figyeljük meg, mi változott a summary kimenetében!
Mit jelent a 10 és 11 szám?
Adjunk egy type nevű csúcsjellemzőt a hálózathoz! Az egyes csúcsok esetén az értéke 0 vagy 1 legyen.
Írja le, most mi változott a summary által kiírt adatokban!

Megoldás

Az első sor végén megjelent a gráf name jellemzője.
A jellemzőlista kibővült két taggal:
date (g), name (g)
A hálózatnak 10 csúcsa és 11 éle van.
Például:

s.vs["type"] = [0,1,1,0]

A jellemzőlista kibővült a type (v) taggal, és megjelent a T az első sorban: UNWT-t értéket mutat a négy flag.

Megjegyzések:
A gráfjellemzőket, mint látjuk, (g) betű jelzi.
A csúcs és élszám, valamint a gráf neve közötti -- csak elválasztó jel.

Jellemzők törlése

Tetszőleges jellemzőt törölhetünk a del utasítással.
Töröljük például a csúcsok méreteit:

```
del s.vs["size"]
```

Él-, csúcs- és gráfjellemzők listázása

Az Graph, a VertexSeq és az EdgeSeq osztálybeli objektumoknak van attributes metódusuk, amellyel listázhatóak a jellemzőik nevei az alábbi módon.

```
net.attributes()
net.vs.attributes()
net.es.attributes()
```

Ilymódon az is vizsgálható, hogy egy jellemző létezik-e:

```
if "size" not in net.vs.attributes():
    net.vs["size"] = 13
```

Adott jellemzőjű csúcsok/élek kiválogatása

A VertexSeq és EdgeSeq osztály objektumainak van egy select nevű metódusa, amely többféle lehetőséget ad a csúcsok kiválogatására.
Ha egy adott jellemzőnek adott értéknek kell lennie, akkor azt kulcsszavas argumentummal szűrhetjük ki.

```
females = s.vs.select(gender="f")
```

Ilyenkor egy újabb VertexSeq-objektumot kapunk, amely az s.vs-sel szemben csak a csúcsok egy részhalmazát tartalmazza, de ahhoz hasonló módon kezelhető.
Megnézhetjük például a name jellemzőiket, vagy beállíthatunk valamilyen tulajdonságukat (pl. nyugdíjkorhatárt):

```
females["name"]
# értéke ['a', 'c', 'e', 'g', 'i']
females["retirement"] = 60
```

Feladat

Válogassa ki az s szociális hálózat élei közül azokat, amelyeknek a súlya 2 (vagy valamilyen más létező súly, amiből lehetőleg több is van)!
Változtassa meg ezeknek az éleknek a színét narancssárgára ("orange")!
Vizsgálja meg az s hálózat éleinek színe változott-e!

Megoldás

```
two = s.es.select(weight=2)
two["color"] = "orange"
s.es["color"]   # Ebben szerepel az orange szín.
```

Látjuk tehát, hogy ha a select-tel kiválasztott VertexSeq-objektumot módosítjuk, az az eredeti hálózatot módosítja.
De mi lesz a többi csúcs (a férfiak) esetén a "retirement" értéke? Azoknál nem állítottunk be értéket.

```
s.vs["retirement"]
# Ezt kapjuk: [60, None, 60, None, 60, None, 60, None, 60, None]
```

Látható, hogy a be nem állított értékek helyén None áll.

Feladat

A már említett módon kívül (szótár + listaértelmezés) így is beállíthatjuk például az összes hölgyhöz tartozó színt.
Állítsuk át az összes nőt jelölő csúcs színét "magenta" értékre és ellenőrizze kirajzoltatással a kapott hálózatot!

Megoldás

```
females["color"] = "magenta"
```

Szűrés adott küszöbértékre

Gyakran nem adott értéket kell keresnünk, hanem valamilyen határ feletti vagy alatti értékeket kell megkeresnünk. A select kulcsavához ilyenkor hozzá kell fűzni aláhúzás karakterrel a lt, le, gt, ge szavak valamelyikét, ezek sorban a nagyobb, nagyobb egyenlő, kisebb, kisebb egyenlő relációknak felelnek meg (less than, less or equal, greater than, greater or equal). Például ha a 2-es vagy kisebb súlyú éleket fehérrel szeretnénk ábrázolni, vagy törölni szeretnénk, akkor előbb ki kell válogatni azokat:

```
weak = s.es.select(weight_le=2)
len(weak)    # Hány ilyen él van?
weak["color"] = "white"
```

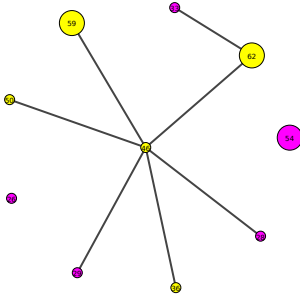
Mivel az élek színe a háttér színével egyezik, csak akkor vesszük észre a létezésüket, ha másik éleket eltakarnak.

Feladat

Tárolja el az egyes személyek korát évben a csúcsok "age" jellemzőjében! Ezek az indexelés sorrendjében 26, 44, 54, 62, 33, 59, 29, 36, 28, 46.
Növelje meg az 50 évnél idősebbek csúcsok méretét 50-re! (Számoltassa meg hány ilyen van, ezzel ellenőrizve!)
Rendezze úgy, hogy a csúcsok mellé a kor legyen kiírva!
Ellenőrizze kirajzoltatással, hogy helyesen dolgozott!

Megoldás

```
s.vs["age"] = [26, 50, 54, 62, 33, 59, 29, 36, 28, 46]
old = s.vs.select(age_gt=50)
len(old)    # 3-nak kell lennie
old["size"] = 50
s.vs["label"] = s.vs["age"]
igraph.plot(s)
```



Több tulajdonság vizsgálata egyszerre

Ha egyszerre több kulcsszavas argumentumot adunk meg a select metódusnak, akkor azokat a csúcsokat szűri, amely mindegyik feltételnek megfelel.

Feladat

Válassza ki a 20 és 60 év közötti férfiakat (beleértve a határokat)!
Állítsa a színűket sötétkékre ("darkblue")!

Megoldás

A 20 és 60 év közötti férfiak bekékítése:

```
midmen = s.vs.select(age_ge=20, age_le=60, gender="m")
midmen["color"] = "darkblue"
```

Adott tulajdonságú élek törlése

Hálózatok vizsgálatánál szükség lehet arra, hogy csak a kapcsolatok egy részét vegyük figyelembe. Például a gyenge kapcsolatokat el kell hanyagolni, hogy használhatóbb eredményt kapjunk. Az előző leckében megismert delete_vertices és delete_edges metódusok argumentuma lehet VertexSeq illetve EdgeSeq típusú objektum. Törölhetjük azokat az éleket, amelyet az előbb fehérre állítottunk a következő módon:

```
s.delete_edges(weak)
```

(Innentől a weak változót ne használjuk már semmire, mert elvesztette értelmét! Mivel éleket töröltünk, ha valamilyen élsorozatra szükségünk van, inkább kérdezzük le újra, ne használjuk a régieket, hacsak nem értjük pontosan, mit teszünk!)

Vázlat

Lecke: A GraphML formátum, szomszédok, részgráfok, gyakorlás

Tervezett időtartam 90 perc.

Követelmények: A hallgató

- fel tudja sorolni a GraphML formátum 3 jellemzőjét,
- tudjon betölteni GraphML formátumú fájlból hálózatokat,
- tudjon szűrni adott csúcs ki- és beszomszédaira,
- tudja meghatározni egy csúcs szomszédait (irányítottnál ki- és beszomszédokat),
- létre tudjon hozni részgráfokat csúcsok indexét, VertexSeq típusú objektumok illetve adott tulajdonságok alapján,
- képes legyen elmenteni GraphML formátumba hálózatokat.

Bevezetés

Ebben a leckében gyakorolni fogjuk a korábban megismert metódusok használatát nagy hálózat esetén. Mivel ezeket a hálózatokat nehéz lenne kézzel létrehozni, meg kell ismerkednünk azzal, hogyan lehet hálózatokat fájlból betölteni. A leckében használt hálózat több tízezer csúcsot tartalmaz, ezért nem lesz mód rá, hogy a korábbihoz hasonló módon kirajzoltassuk. Megtanuljuk, hogyan lehet csak valamely részletét kirajzoltatni: ehhez részgráfokat kell tudni létrehozni. A lecke végére ki fogunk tudni rajzoltatni egy csúcsot a szomszédaival.

A GraphML formátum

Az igraph több fájlformátumból képes hálózatot betölteni és azokba kiírni. Mi a GraphML-t fogjuk használni, amelyről a következőket kell tudni:

- a formátuma XML alapú,
- alkalmas irányított, irányítatlan és vegyes gráfok tárolására,
- tárolhatóak gráf-, csúcs- és éljellemzők benne.

Hálózatok betöltése

A továbbiakban nagyobb hálózatokat fogunk vizsgálni, ehhez szükségünk lesz a hálózatok betöltésére. A Graph osztály Read osztálymetódusa képes a kiterjesztésből meghatározni a fájl típusát, és betölteni. A hálózat abszolút vagy relatív útvonalát kell megadni argumentumként:

```
net = igraph.Graph.Read("/home/bb/halozatok/x.graphml")
net = igraph.Graph.Read("halozatok/x.graphml")
```

(A Python3-mal a tömörített fájlformátumok (pl. GraphMLz) olvasása és írása nem működik az igraph 0.6-ban.)

Tipp: Ha a fájl neve hosszú, akkor a kiegészítéséhez érdemes a Tab-ot használni.

Feladat

Az elearning oldalról a jelenlegi modultól letölthető egy nagy hálózat, a Debian szoftvercsomagjainak a hálózata. (Majd később lesz szó róla, mi ez, most az a lényeg, hogy csomó jellemzővel rendelkezik.)
Töltse le a hálózatot és töltse be deb néven!
Határozza meg hány csúcsjellemző, éljellemző, gráfjellemző van a gráfban!
Határozza meg, mennyi olyan csúcs van, ahol a filesize jellemző nagyobb, mint 4000!
Ez hány százaléka az összes csúcsnak?

Megoldás

```
import igraph
deb = igraph.Graph.Read("debian-7.3-packages-2014-04-30_09.03GMT.graphml")

igraph.summary(deb)
# 13 gráfjellemző, 10 csúcsjellemző és 1 éljellemző van

bigfiles = deb.vs.select(filesize_gt=4000)
len(bigfiles)
# 35765 ilyen fájl van
# ha kettővel többet kapott, a pont 4000 bájtosokat is belevette

len(bigfiles) / deb.vcount()
# Nagyjából a fájlok 85 százaléka nagyobb 4000 bájtnál.
```

Feladatok

Határozza meg milyen értékei vannak a csúcsok típusainak (type)!
Határozza meg, a csúcsok mekkora része 0-ás típusú!
Írassa ki a hálózat frissítésének dátumát (update_time)!

Megoldás

```
set(deb.vs["type"])
# A típus 0.0 vagy 1.0 lehet.
# Eredetileg 0 és 1,
# de a graphml formátumból beolvasáskor mind lebegőpontos lett.

zerotype = deb.vs.select(type=0)
# 0.0 helyett 0 is működik, mivel 0 == 0.0

len(zerotype) / deb.vcount()
# Kb. 87 és fél százalék

print(deb["update_time"])
# 2014-04-30 09:03:35 GMT (azaz Greenwich-i idő szerint ennyi)
```

Feladat

A 0 típusú csúcsok hány százalékának nagyobb a fájlmérete, mint 4000 bájt?

Megoldás

```
big0 = deb.vs.select(filesize_gt=4000, type=0)
len(big0) / len(zerotype)
# 97.4% nagyobb 4000-nél a nullás típusúak közül.
```

Számonkérés során nem kerül elő az alábbi, de hasznos lehet. A select metódus VertexSeq típusú objektummal tér vissza, amely tovább szűrhető ugyanúgy, mint a deb.vs (az is a VertexSeq osztály objektuma). Ilymódon a big0 csúcssorozatot a következő módokon is megkaphattuk volna:

```
bigfiles  = zerotype.select(filesize_gt=4000)
big0  = bigfiles.select(type=0)
```

A Debian szoftvercsomagjának hálózata több mint negyvenezer csúcsból áll. Az ilyen nagy hálózatokat nehéz kirajzoltatni úgy, hogy az megmutasson valami lényegeset hálózatról, és ne csak egymást részben eltakaró csúcsok, és követhetetlen élek kusza ábrája legyen. Ezért csupán azt célozzuk meg, hogy a hálózat egy részét rajzoltatjuk ki: ebben a leckében egy csúcsot a szomszédaival, később egy kisebb komponenst.

Szomszédság kirajzolása

A következő lépésekben egy csúcs szomszédságának a kirajzolása lesz a cél. Ezt a következő lépésekben tesszük meg:

1. Meghatározzuk a csúcs szomszédainak indexeit a (neighbors metódus).
2. Meghatározzuk a csúcs indexét, és hozzáadjuk az indexlistához (find metódus).
3. Létrehozzuk az indexek alapján a részhálózatot (subgraph metódus)
4. Kirajzoltajuk a részgráfot (plot függvény).

Szomszédok meghatározása

A csúcs szomszédait a Graph osztály neighbors metódusával kereshetjük meg. Ez a szomszédok indexeinek a listáját adja vissza. Az egyetlen kötelező paramétere lehet egy csúcs indexe, vagy, amennyiben van name csúcsjellemző, a csúcs neve.
Megkereshetjük például a "vim" nevű csomag szomszédait a következő módon:

```
vim_nb = deb.neighbors("vim")
```

Csúcs megkeresése

Egy csúcsot a neve alapján a VertexSeq osztály find metódusával keressük meg, majd hozzáadjuk az indexét a csúcslistához:

```
vim = deb.vs.find("vim")
vim_nb.append(vim.index)
```

Másik lehetőség

A find metódus egy igraph.Vertex típusú objektummal tér vissza. Ezt a neighbors metódus szintén elfogadja paraméterként. Az előző két oldalon leírtakat az alábbi módon is megcsinálhattuk volna:

```
vim = deb.vs.find("vim")
vim_nb = deb.neighbors(vim)
vim_nb.append(vim.index)
```

Miután eljut oda, hogy az első módon kirajzoltatta a szomszédságot, érdemes kipróbálni ilyen úton is.

Részgráf létrehozása

A hálózat egy részének létrehozására az igraph-ban a Graph osztály subgraph metódusa ad rugalmas lehetőséget. Argumentumként megadhatunk neki indexlistát, vagy VertexSeq típusú objektumot is.
A létrehozott részhálózat örökölni fogja az eredeti hálózat csúcs- és éljellemzőit.

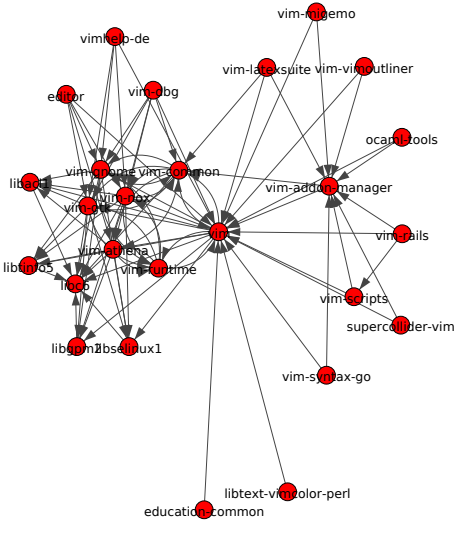
A részgráf létrehozása és kirajzoltatása

Mivel szeretnénk látni a csúcsok neveit, érdemes létrehozni a label csúcsjellemzőt az eredeti gráfban a name csúcsjellemzőből.
A vim nevű csúcsból (csomagból) és szomszédaiból álló részgráfot következő módon hozhatjuk létre és rajzoltathatjuk ki:

```
deb.vs["label"] = deb.vs["name"]
vim_nb_net = deb.subgraph(vim_nb)
igraph.plot(vim_neighbors, margin=(90,20)*2)
```

A kirajzoltatáskor megadtuk a margin argumentumot is, hogy a feliratok ne lógjanak ki: jobb és baloldalt 90 pixel helyet hagytunk a szélső csúcsok középpontjától, fent és lent 20-at.

A kapott ábra



Miért az eredetiben hoztuk létre az "label"-t?

Létrehozhattuk volna a részgráfban is a "label" cimkét. Ha egyetlenegyszer hozunk létre részgráfot, ez gazdaságos is. (Az az előnye is megvan, hogy kevesebb memóriát igényel.)

Ha viszont több részgráfot szeretnénk létrehozni és ábrázolni, akkor egyetlenegyszer elég létrehozni a "label" cimkét az eredeti gráfban, és onnan az összes részgráf örökli.

Feladat

Hozzon létre olyan programot, amely betölti a hálózatot, és kirajzolja egy fájlba az adott nevű csomagból és szomszédaiból álló hálózat ábráját! A fájl nevében legyen benne a csomag neve!
Rajzoltassa ki az mc nevű csomag szomszédságát!
Törekedjen rá, hogy a programkód áttekinthető legyen!
Használhat függvény(eke)t vagy osztály(oka)t is.

Bevezetés a megoldáshoz

A továbbiakban egy olyan megoldás olvasható, amelyben egy Neighborhood osztály felelős a részgráf létrehozásáért és kirajzolásáért. Természetesen tisztán függvényekkel is meg lehetett volna oldani a feladatot, de akkor vagy egy nagy hosszú függvényt kellett volna használni, ami nehezen átlátható és tesztelhető, vagy sok paramétert átadogatni a függvények között.
Először magát az osztályt mutatom be, majd a többi részt a programból.

```
class Neighborhood:
    def __init__(self, network, vertex_name):
        self.network = network
        self.vertex_name = vertex_name
        self.vertex = network.vs.find(vertex_name)
        self.vertex_index_list = self.collect_vertices()
        self.subgraph = self.create_subgraph()

    def collect_vertices(self):
        vertices = self.network.neighbors(self.vertex)
        vertices.append(self.vertex.index)
        return vertices

    def create_subgraph(self):
        create_labels(self.network)
        return self.network.subgraph(self.vertex_index_list)

    def plot(self):
        igraph.plot(self.subgraph,
                "{0}_neighborhood.pdf".format(self.vertex_name),
                margin=(90,20)*2, bbox=(800,600))
```

A második és harmadik metódus az __init__ metódust szolgálja ki, hogy az létrehozhassa a részgráfot, és az utolsó metódus rajzolja ki azt. A kirajzolásnál a bbox kulcsszavas argumetnum megadja, hogy a 600x600-as alapértelmezett érték helyett 800x600-as legyen az ábra mérete.
Természetesen importálni kell a program elején az igraph metódust.
A cimkék létrehozása, mivel nem csak a szomszédságnál lehet hasznos, külön függvénybe került.
A függvény további része látható a következő oldalon.

```
#!/usr/bin/env python3

import igraph

def create_labels(network):
    if not "label" in network.vs.attributes():
        network.vs["label"] = network.vs["name"]

class Neighborhood:
    ...

if __name__ == '__main__':
    deb = igraph.Graph.Read(
            "debian-7.3-packages-2014-04-30_09.03GMT.graphml")
    mc_neighborhood = Neighborhood(deb, "mc")
    mc_neighborhood.plot()
```

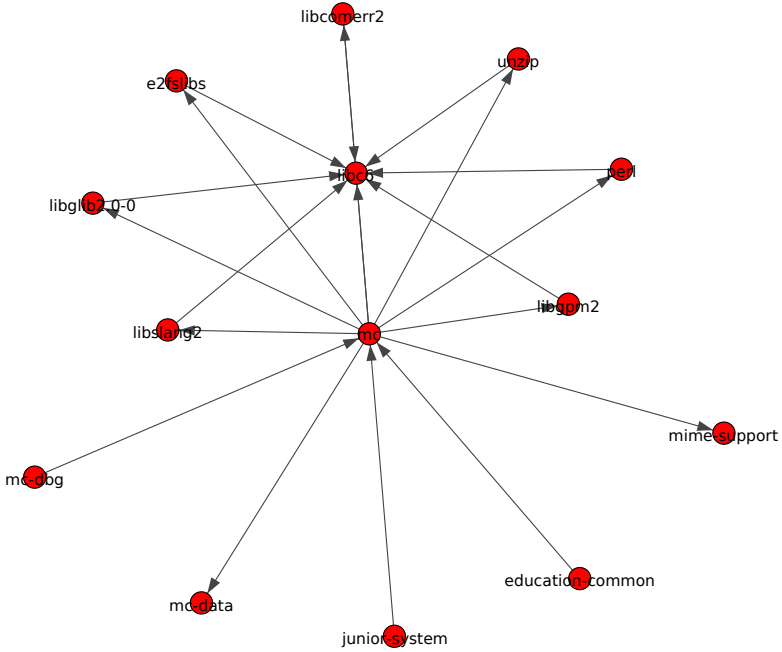
Végső megjegyzések

Az előző oldal az egész programot tartalmazza a Neighborhood osztály kivételével. Annak csak a helye van feltüntetve, és . . . jelöli a kihagyott részt.
A programot vagy futtathatóvá tesszük és meghívjuk közvetlenül (csak ekkor kell az első sor), vagy a python3 paranccsal hívjuk meg,

```
chmod +x neighborhood.py
neighborhood.py
```

```
python3 neighborhood.py
```

vagy az IDLE3-ban futtatjuk le. A kapott ábrát a következő oldal tartalmazza.


libcomerr2
unzip
e2fslibs
libc6
perl
libglib2.0-0
libgpm2
libslang2
mc
mime-support
mc-obg
education-common
mc-data
junior-system

Feladat

Jó lenne, ha az ábra mérete nem lenne belekódolva megváltoztathatatlanul a plot metódusba, hanem mi adhatnánk meg kulcsszavas argumentumokat a metódusnak valahogy így:

```
mc_neighborhood.plot(bbox=(1000,800), edge_color="orange")
```

Keressen elegáns megoldást erre! Akár a saját változatát is módosíthatja ennek megfelelően.

Egyéni megoldásokat szívesen látok a elearning oldal hallgatói forumában.

Egy lehetséges megoldás, és egy másik

```
def plot(self, **kwargs):
    igraph.plot(self.subgraph,
            "{0}_neighborhood.pdf".format(self.vertex_name),
            **kwargs)

def plot(self, **given_kwargs):
    kwargs = dict(margin=(90,20)*2, bbox=(800,600))
    kwargs.update(given_kwargs)
    igraph.plot(self.subgraph,
            "{0}_neighborhood.pdf".format(self.vertex_name),
            **kwargs)
```

A második megoldás egyesíti azt, hogy normális esetben a kellemesebb értékekkel dolgozik anélkül, hogy meg kellene adnunk a paramétereket, és mégis bármelyik kulcsszavas paramétert megváltoztathatjuk kívülről.

Be- és kiszomszédok

Irányított hálózatoknál megkülönböztethetünk olyan szomszédokat, amelyből befelé mutatnak az élek (beszomszédok), és olyanokat is, amelyekbe kifelé mutatnak (kiszomszédok). Ezt az neighbors függvény mode argumentumával választhatjuk ki, hogyha csak az egyiket akarjuk listázni: értéke "in" és "out" lehet (illetve alapesetben "all").
A vim csúcs beszomszédainak indexeit például meghatározhatjuk így:

```
vim_in = deb.neighbors("vim", mode="in")
```

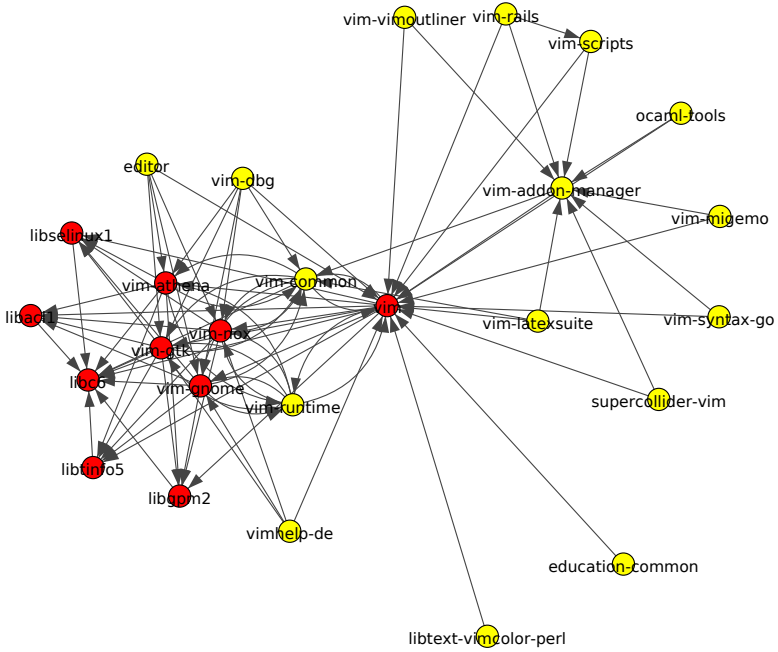
Feladat

Színezzük a vim csomag beszomszédait sárgára, de csak a vim_nb_net részhálózatban, az eredeti deb hálózat színeit ne piszkáljuk! Rajzoltassuk ki újra!

Megoldás

```
vim_nb_net.neighbors("vim", mode="in")
inneighbors = vim_nb_net.neighbors("vim", mode="in")
vim_nb_net.vs.select(inneighbors)["color"] = "yellow"
igraph.plot(vim_nb_net, margin=(90,20)*2)
```

(Az kapott csúcsindexek mások a részgráfban, mint az eredeti gráfban kerestük volna.)
Az ábra következő oldalon látható.


vim-vimoutliner
vim-rails
vim-scripts
ocaml-tools
editor
vim-dbg
vim-addon-manager
vim-migemo
libselinux1
vim-athena
vim-common
vim
libacl1
vim-nox
vim-latexsuite
vim-syntax-go
vim-gtk
libc6
vim-gnome
vim-runtime
supercollider-vim
libtinfo5
libgpm2
vimhelp-de
education-common
libtext-vimcolor-perl

Kitekintés

Ebben a fejezetben megismerkedtünk a szomszédokat listázó neighbors metódussal, amivel létrehoztuk egy csúcs szomszédságát tartalmazó részhálózatot. Ugyanezt egyszerűbben megtehettük volna a neighborhood metódussal is de úgy gondolom, hogy az előbbi függvény felhasználási köre tágabb, ezért csak ez képezi a tananyag részét. Mindazonáltal ha egy csúcs tágabb környezetét szeretné valaki tanulmányozni vagy kirajzoltatni, akkor érdemes az utóbbi metódust is megismerni.

Vázlat

Lecke: Komponensek és átmérő meghatározása az igraph használatával

Tervezett időtartam 90 perc.

Követelmények: A hallgató

- meg tudja határozni irányítatlan hálózatok komponenseinek számát, azok méreteit (csúcsainak számát),
- meg tudja határozni irányított hálózatok erősen és gyengén összefüggő komponenseinek számát és azok méreteit,
- ki tud rajzoltatni egy komponenst,
- meg tudja határozni egy hálózatban a legnagyobb komponens relatív méretét az egész hálózathoz képest,
- meg tudja határozni irányítatlan hálózat átmérőjét,
- meg tudja határozni irányítatlan hálózatban két csúcs távolságát,
- meg tudja határozni és értelmezni irányított hálózat kétféle átmérőjét.

Komponensek ismétlés

Az kurzus elején megismerkedett a komponensekkel. Megtanulta, hogy az irányított hálózatok esetén kétféle komponenst értelmezünk. A gyengén összefüggő komponensnél eltekintünk az élek irányától, és ugyanúgy vizsgáljuk a komponenseket, mintha irányítatlan lenne. Az erősen összefüggő komponensek esetén csak akkor sorolunk egy komponensbe két csúcsot, ha közöttük oda-vissza el tudunk jutni a nyilak irányába.

A hálózatok csúcsainak a számát röviden a *hálózatok méretének* nevezzük ebben a kurzusban.

Komponensek meghatározása igraph-ban

A komponenseket a Graph osztály components metódusával határozhatjuk meg. Ez alkalmas az irányítatlan hálózatok komponenseinek meghatározására, valamint az irányított hálózatok erős és gyenge komponenseinek a meghatározására.
Az irányított esetben alapesetben erősen összefüggő komponenseket keres, gyenge egy mode nevű kulcsszavas argumentummal meg kell adni:

```
net.components(mode="weak")
```

Minden esetben egy VertexClustering típusú objektumot ad vissza a metódus, aminek a metódusaival minden fontos dolgot meg lehet tudni a komponensekről.

A komponensek méretei

A VertexClustering osztály amilyen sorrendben megtalálja az egyes komponenseket, olyan sorrendben tárolja a hozzá tartozó csúcsok indexeit. Az osztály sizes metódusa az egyes komponensek méreteivel tér vissza a megtalálás sorrendjében, tehát nincsenek rendezve csúcsszámok szerint a komponensek. (Bár – ha a csúcsok nagy része egy komponensbe tartozik – nagy a valószínűsége, hogy hamar megtalálja, így a nagy (sok csúccsal rendelkező) komponensek általában az elején vannak.)
A sizes által visszaadott listából sokminden megtudható a komponensekről: a komponensek száma, a legnagyobb és legkisebb komponensek mérete (vagy részletesebben tekintve a komponensek méreteloszlása).

Feladat

Töltse be az előző leckében használt Debian-szoftvercsomagok hálózatát deb néven, hozza létre a gyengén összefüggő komponenseket. Írja ki a sizes metódus által visszaadott lista első 30 elemét!

Megoldás

Az alábbi kóddal létrehozhatóak a gyengén összefüggő komponensek, amelyet az angol rövidítés alapján (weakly connected components) a wcc változónéven tároljuk. A következő sor megadja az első 30 komponens méreteit.

```
deb = igraph.Graph.Read(
      "debian-7.3-packages-2014-04-30_09.03GMT.graphml")
wcc = deb.components(mode="weak")
wcc.sizes()[:30]
# vagy két lépésben:
sizes = wcc.sizes()
sizes[:30]
# Az eredmény [40264, 10, 4, 4, 2, 2, 2, 23, 2, 2, 2, 2, 2, 2, 2, 8, 3,
   2, 2, 8, 2, 2, 3, 2, 2, 6, 2, 2, 6, 2]
```

A méretlista értelmezése

Korábban kiirattuk a gyengén összefüggő komponensek méretlistájának az elejét.

```
sizes[:30]
# Az eredmény [40264, 10, 4, 4, 2, 2, 2, 23, 2, 2, 2, 2, 2, 2, 2, 8, 3,
   2, 2, 8, 2, 2, 3, 2, 2, 6, 2, 2, 6, 2]
```

Az eredmény azt jelenti, hogy a 0 indexű komponens 40264 csúcsot, az 1 indexű 10 csúcsot tartalmaz és így tovább. Ez a továbbiakban majd fontos lesz.

Feladat

Állapítsa meg hány gyengén összefüggő komponenset tartalmaz a Debian-szoftvercsomagok hálózata, és a legnagyobb mekkora hányadát (hány százalékát) tartalmazza a teljes hálózat csúcsainak!

Segítség

A sizes metódus listával tér vissza, aminek a hosszát és maximumát beépített Python-függvényekkel meghatározhatja.

Megoldás

Mivel a sizes által visszaadott listában minden egyes komponensnek megtalálható a mérete, a lista hossza a komponensek számát adja meg. A max függvénnyel pedig megkapható a listából a legnagyobb komponens mérete.

```
wcc = deb.components(mode="weak")
sizes = wcc.sizes()
len(sizes)   # 1501 gyengén összefüggő komponens van.
max(sizes)   # 40264 csúcs tartozik a legnagyobb komponenshez.
max(sizes) / deb.vcount()
  # kb. a csúcsok 96%-a a legnagyobb komponenshez tartozik.
```

Feladat

Csak a sizes listából, a vcount használata nélkül is megkapható a teljes hálózat csúcsszáma, és így megkapható csak a sizes listából a legnagyobb komponens egész hálózathoz viszonyított mértéke is.

A megoldást nem tartalmazza a segédlet.

Feladat

Hozzon létre olyan number_of_components nevű függvényt, amely irányított hálózat esetén visszadja a gyengén és erősen összefüggő komponensek számát két elemű tuple-ben vagy listában!

Egy lehetséges megoldás

```
def number_of_components(net):
    wcc = net.components(mode="weak")
    scc = net.components()
    return [len(cc.sizes()) for cc in (wcc, scc)]
```

Feladat

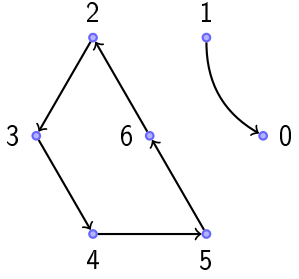
Ha elvégeznénk az alábbi utasításokat mi lenne az egyes utasítások után a gyengén összefüggő illetve az erősen összefüggő komponensek száma?

Határozza meg először anélkül, hogy végrehajtaná, csupán lerajzolva a hálózat egyes állapotait!

Hajtsa végre az utasításokat, és ellenőrizze a megismert függvényekkel, hogy jól gondolta-e!

```
net = igraph.Graph([(1,0), (2,3), (3,4), (4,5), (5,6), (6,2)],
                   directed=True)
net.add_edge(0,1)
net.add_edge(5,0)
net.add_edge(1,2)
net.delete_edges([(5,6)])
net.delete_edges([(4,5)])
net.add_edge(4,6)
```

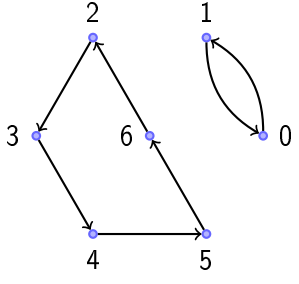
Megoldás



Komponensek száma
gyengén összefüggő: 2
erősen összefüggő: 3

```
net = igraph.Graph([(1,0), (2,3), (3,4), (4,5), (5,6), (6,2)],
                   directed=True)
```

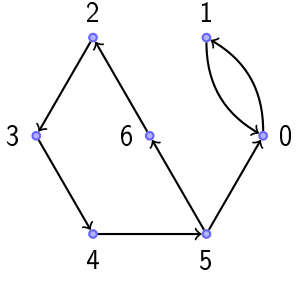
Megoldás



Komponensek száma
gyengén összefüggő: 2
erősen összefüggő: 2

```
net = igraph.Graph([(1,0), (2,3), (3,4), (4,5), (5,6), (6,2)],
                   directed=True)
net.add_edge(0,1)
```

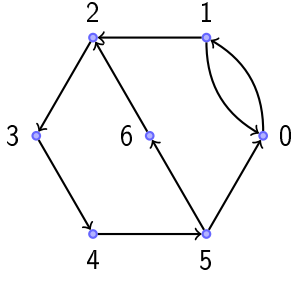
Megoldás



Komponensek száma
gyengén összefüggő: 1
erősen összefüggő: 2

```
net = igraph.Graph([(1,0), (2,3), (3,4), (4,5), (5,6), (6,2)],
                   directed=True)
net.add_edge(0,1)
net.add_edge(5,0)
```

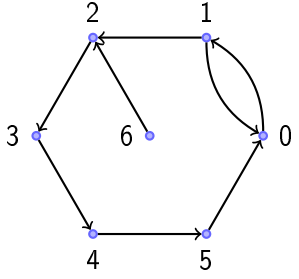
Megoldás



Komponensek száma
gyengén összefüggő: 1
erősen összefüggő: 1

```
net = igraph.Graph([(1,0), (2,3), (3,4), (4,5), (5,6), (6,2)],
                   directed=True)
net.add_edge(0,1)
net.add_edge(5,0)
net.add_edge(1,2)
```

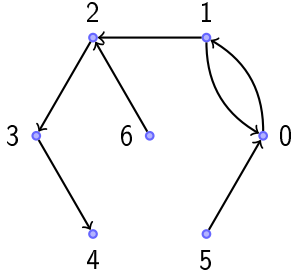
Megoldás



Komponensek száma
gyengén összefüggő: 1
erősen összefüggő: 2

```
net = igraph.Graph([(1,0), (2,3), (3,4), (4,5), (5,6), (6,2)],
                   directed=True)
net.add_edge(0,1)
net.add_edge(5,0)
net.add_edge(1,2)
net.delete_edges([(5,6)])
```

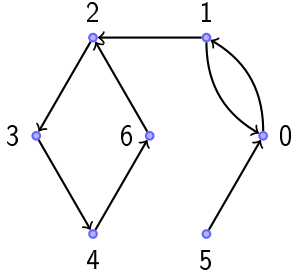
Megoldás



Komponensek száma
gyengén összefüggő: 1
erősen összefüggő: 6

```
net = igraph.Graph([(1,0), (2,3), (3,4), (4,5), (5,6), (6,2)],
                   directed=True)
net.add_edge(0,1)
net.add_edge(5,0)
net.add_edge(1,2)
net.delete_edges([(5,6)])
net.delete_edges([(4,5)])
```

Megoldás



Komponensek száma
gyengén összefüggő: 1
erősen összefüggő: 3

```
net = igraph.Graph([(1,0), (2,3), (3,4), (4,5), (5,6), (6,2)],
                   directed=True)
net.add_edge(0,1)
net.add_edge(5,0)
net.add_edge(1,2)
net.delete_edges([(5,6)])
net.delete_edges([(4,5)])
net.add_edge(4,6)
```

Egy komponens kirajzoltatása, 1. módszer

Kétféleképpen fogunk kirajzoltani komponens(eke)t. Az első módszer egy komponens kirajzolására alkalmas gyorsabb módszer. A másodikkal akár több komponenst is kirajzoltathatunk egyszerre!
Az első módszer lépései:

1. Hozzuk létre a komponenseket!
2. Keressük meg a kirajzoltatandó komponens indexét és hozzuk létre a komponenst tartalmazó részgráfot!
3. Rajzoltassuk ki a részgráfot a már ismert módon!

A komponensek létrehozásával és a hálózatok kirajzolásával már foglalkoztunk, tehát egyedül a második lépést kell megbeszélni.

2. lépés: A részgráf létrehozása

A VertexClustering osztálynak van egy subgraph metódusa, amelynek a komponens indexét megadva az annak megfelelő részgráfot hozza létre a csúcsai között vezető összes éllel.

Megkereshetünk például egy adott méretű komponens indexét, ha a komponensek méretét tartalmazó lista index metódusát használjuk. Természetesen ilyenkor egyet kapunk az azonos méretűek közül, a legkisebb indexűt.

Feladat

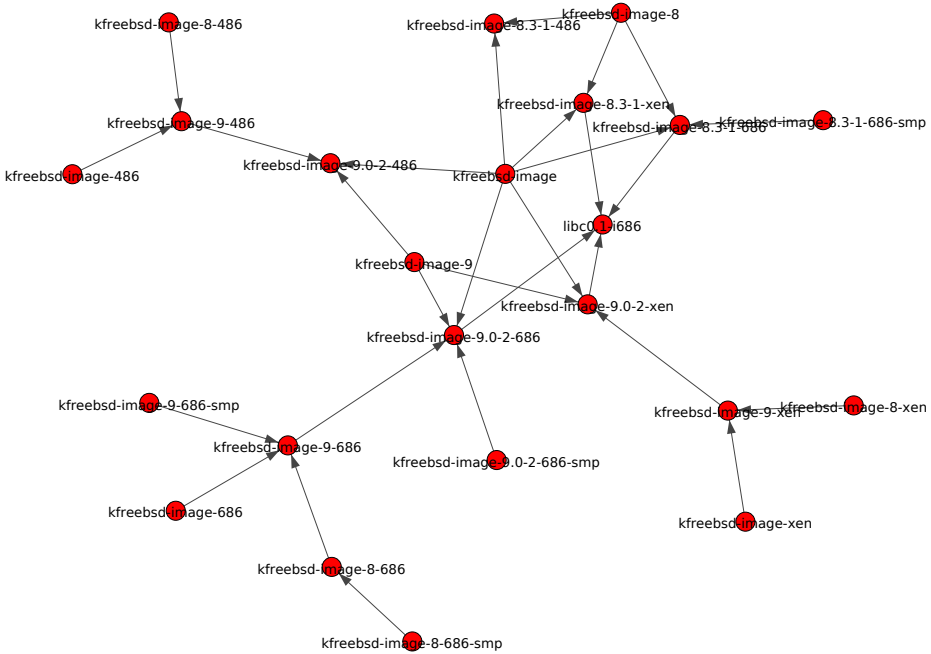
Létrehoztuk már a deb hálózat gyengén összefüggő komponenseit, és a komponensek méretét tartalmazó sizes listát.

Keressünk meg a 23 méretű komponens indexét, majd hozzuk létre a részgráfot! Határozzuk meg a részgráf éleinek a számát és ábrázoljuk a csúcsok nevével!

Megoldás

```
ix = sizes.index(23)
sub23 = wcc.subgraph(ix)
sub23.ecount()    # 29 él van benne.
sub23.vs["label"] = sub23.vs["name"]  # Ha nem tettük az eredeti gráfra.
igraph.plot(sub23, bbox=(1000,700), margin=(90,20,90,20))
```

A bbox és margin kulcsszavas argumentumok nem kötelezőek, de így kevésbé fednek át illetve kevésbé lógnak ki a feliratok.
A kapott ábra a következő oldalon látható.


kfreebsd-image-8-486
kfreebsd-image-8.3-1-486
kfreebsd-image-8
kfreebsd-image-8.3-1-xen
kfreebsd-image-9-486
kfreebsd-image-8.3-1-686
kfreebsd-image-8.3-1-686-smp
kfreebsd-image-486
kfreebsd-image-9.0-2-486
kfreebsd-image
libc0.1-i686
kfreebsd-image-9
kfreebsd-image-9.0-2-xen
kfreebsd-image-9.0-2-686
kfreebsd-image-9-686-smp
kfreebsd-image-9-xen
kfreebsd-image-8-xen
kfreebsd-image-9-686
kfreebsd-image-9.0-2-686-smp
kfreebsd-image-686
kfreebsd-image-xen
kfreebsd-image-8-686
kfreebsd-image-8-686-smp

A legnagyobb komponens kirajzoltatása

A legnagyobb komponenst gyakran óriáskomponensnek is szokták nevezni. A VertexClustering objektumoknak a giant metódusa előállítja a legnagyobb komponens részgráfját.
Mint láttuk, a legnagyobb gyengén összefüggő komponens a csúcsok 96%-át, több, mint 40000 csúcsot, foglal magába a Debian-szoftvercsomagok hálózatában, amit nem érdemes kirajzoltatni.
A legnagyobb gyengén összefüggő komponens viszont csak 54 csúcsból áll, ami jól kirajzolható.

Feladat

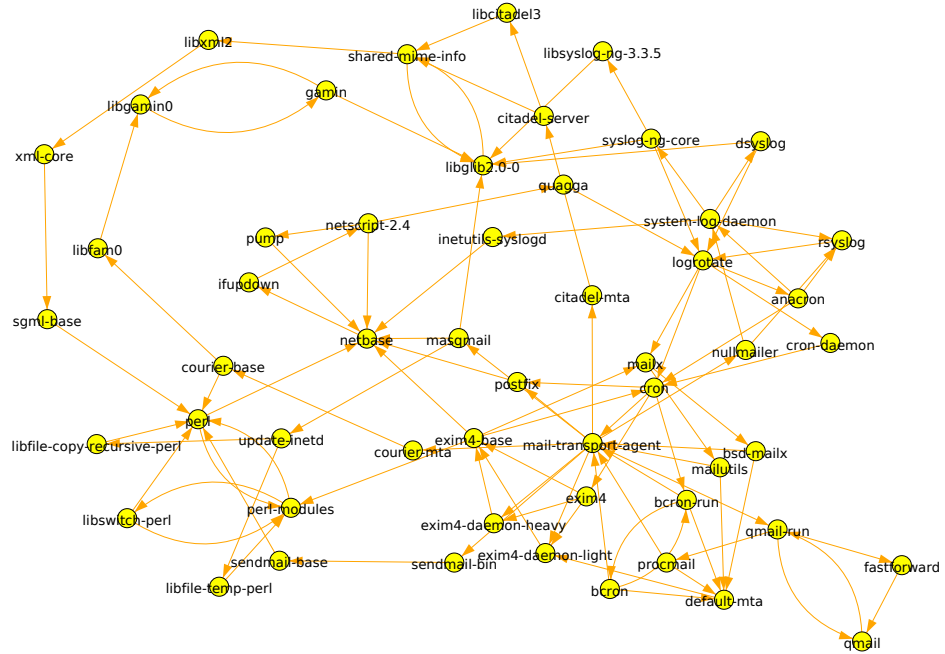
Rajzoltassa ki a deb hálózat legnagyobb erősen összefüggő komponensét úgy, hogy jól látsszanak a feliratok!

Egy megoldási lehetőség

Hogy a feliratok látsszanak, a csúcsok és élek színét világosabbra vehetjük. A sok felirathoz nagy ábra kell (bbox). A margóval (margin) lehet játszani, hogy a feliratok beleférjenek, ehhez érdemes az elrendezést rögzíteni (layout), de akár módosítani is lehet kicsit a koordinátákon, hogy ne lógjanak egymásra a feliratok.

```
scc = deb.components()
sgiant = scc.giant()
lo=sgiant.layout()
igraph.plot(sgiant, "deb_largest_scc.pdf", layout=lo,
        bbox=(1000,700), margin=(50,20)*2,
        vertex_color="yellow", edge_color="orange")
```

A kapott ábra a következő oldalon található.


libcitadel3
libxml2
shared-mime-info
libsyslog-ng-3.3.5
gamin
libgamin0
citadel-server
xml-core
syslog-ng-core
dsyslog
libglib2.0-0
quagga
netscript-2.4
system-log-daemon
pump
inetutils-syslogd
rsyslog
libfam0
logrotate
ifupdown
citadel-mta
anacron
sgml-base
netbase
masqmail
nullmailer
cron-daemon
courier-base
mailx
postfix
cron
perl
libfile-copy-recursive-perl
update-inetd
courier-mta
exim4-base
mail-transport-agent
bsd-mailx
mailutils
exim4
bcron-run
libswitch-perl
perl-modules
exim4-daemon-heavy
qmail-run
sendmail-base
exim4-daemon-light
sendmail-bin
procmail
fastforward
libfile-temp-perl
bcron
default-mta
qmail

Komponensek kirajzoltatása

A VertexClustering típusú objektumok indexelhetőek: az elsőnek megtalált komponens indexe 0, a következőé 1 és így tovább. Egy ilyen objektumot indexelve a komponens csúcsainak indexlistáját kapjuk.
A gyengén összefüggő komponenseket a wcc változóban tároltuk, és a méretek listájának elejét kiírattuk. A második (1-es indexű) elem értéke 10 volt. Kiírathatjuk ennek a 10 elemű komponens csúcsainak indexeit:

```
wcc[1]
# Ezt kapjuk: [445, 446, 17692, 25707, 26616, 26617, 28656, 32416, 36009, 39164]
```

A lecke vége még nincs készen.