



EÖTVÖS LORÁND TUDOMÁNYEGYETEM
INFORMATIKAI KAR
KOMPUTERALGEBRA TANSZÉK

Fokszorozatokat ellenőrző algoritmusok

TDK dolgozat

Témavezető:
Dr. Iványi Antal Miklós
egyetemi tanár

Készítette:
Lucz Loránd, Sótér Péter
II. évfolyam
Programtervező informatikus MSc

Budapest, 2011

Tartalomjegyzék

1. Bevezetés	3
2. Pontos algoritmusok	4
2.1. Havel-Hakimi algoritmus (HH)	4
2.2. Eltoló Havel-Hakimi algoritmus (HHE)	6
2.3. Erdős-Gallai algoritmus (EG)	6
2.4. Rövidített Erdős-Gallai algoritmus (EGR)	8
2.5. Ugró Erdős-Gallai algoritmus (EGU)	9
3. Közelítő algoritmusok	10
3.1. Gyors teszt (FT)	11
3.2. Paritás teszt (PT)	11
3.3. Binomiális teszt (BT)	12
3.4. Binomiális teszt ugrással (BTJ)	13
4. Leszámlálási eredmények	13
5. Elemzések	17
5.1. Pontos algoritmusok	17
5.2. Közelítő algoritmusok	19
6. Javasolt új algoritmusok	22
6.1. Havel-Hakimi-Eltoló algoritmus gyors teszttel (HHEGy)	22
6.2. Lineáris-Erdős-Gallai-Ugró algoritmus (EGLU)	24
7. Összefoglalás	26
Hivatkozások	27

1. Bevezetés

Napjainkban a minket körülölelő világban megtalálható objektumok és személyek kapcsolata sokféle módon modellezhető. Ennek egy lehetséges módja, hogy az összekapcsolódó tárgyakat és problémákat gráfok segítségével reprezentáljuk, ahogy azt tette Landau 1953-as [21] cikkében, amelyben az állatfajok közti hierarchiát modellezte, egyszerű irányított gráfok segítségével. Cikkében lineáris idő alatt ellenőrizhető szükséges és elégséges feltételt adott arra, hogy egész számok sorozata egy egyszerű irányított gráf fokszorozata legyen.

1955-ben Václav Havel [9] vetette fel a kérdést, hogy hogyan lehetne eldönteni n egész szám monoton csökkenő sorozatáról, hogy az egy n csúcsú egyszerű gráf csúcsainak fokszámaikat tartalmazó sorozat, azaz a gráfhoz tartozó *fokszorozat-e*. Cikkében a kérdésre válaszként meg is adott egy négyzetes futási idejű algoritmust, amely a megadott fokszorozatot nem csak ellenőrizte, hanem a hozzátartozó gráf előállítására is képes volt. Tőle függetlenül 1962-ben Hakimi [8] is publikálta a módszert, ezért az algoritmusra rendszerint Havel-Hakimi algoritmusként hivatkoznak a szakirodalomban. Hakimi cikkében a problémát egy kémiai feladat megoldása kapcsán vetette fel és oldotta meg.

1960-ban Erdős és Gallai [5] megadott egy algoritmust, amely anélkül képes eldönteni a kérdést, hogy előállítaná a fokszorozathoz tartozó gráfot. Azonban ennek az algoritmusnak is négyzetes futási időre van szüksége a kérdés eldöntéséhez.

Az eredeti problémának több más tudományterületen is van megfelelője, amely ugyanennek a kérdésnek az eldöntését teszi szükségessé. Például Liljeros és munkatársai [22] az emberi szokásokat modellezve, Kim és munkatársai [18], valamint Newman és munkatársai [25] hálózati problémákra vonatkozóan, Iványi és munkatársai [10, 11, 13, 16] sportbeli alkalmazások kapcsán hivatkoztak a problémára.

Az eredeti algoritmusok ugyan csak egyszerű gráfokra alkalmazhatóak, azonban a későbbiekben kiterjesztették őket egyszerű irányított gráfokra [6, 19, 20], valamint irányítatlan [4, 26] és irányított [10, 11] multigráfokra is.

Erdős és Gallai módszeréhez 2003-ban Tripathy és Vijay [31] megadtak két javaslatot, amellyel gyorsítani lehet a kérdés eldöntését, de még ez a felgyorsított módszer is négyzetes futási időt igényel.

A *The On-Line Encyclopedia of Integer Sequences* [27] internetes adatbázisban találhatóak adatok arra vonatkozóan, hogy összesen hány olyan n -hosszú monoton csökkenő sorozat létezik, amely *helyreállítható*, azaz megadható hozzá egy olyan gráf, amelynek fokszorozata a megadott sorozat. Ezeknek a sorozatoknak a számát a későbbiekben G_n -nel jelöljük. Az adatbázis $n \leq 23$ -ig tartalmazta ezeket az adatokat. $n = 20$ -tól $n = 23$ -ig például Cohen adta meg ezeket az adatokat 2011. július 9-én. A dolgozatban ismertetett eredmények felhasználásával egy minden szöbajövő sorozatot ellenőrző programot írtunk, és azzal megadtuk a G_n értékét $n = 29$ -ig. Eredményeink megtalálhatóak az OEIS adatbázis A004251-es sorozatában [28], a számításokat végző programot pedig a [23] TDK dolgozatban ismertetjük részletesen.

Dolgozatunk egyik célja, hogy olyan közelítő algoritmusokat adjunk meg, amelyeket felhasználva a szakirodalomból ismert pontos algoritmusok gyorsíthatóak. A gyorsítást úgy értjük, hogy a közelítő algoritmusokkal előteszteljük a sorozatokat, így azokat gyorsabban (kevesebb művelet elvégzésével) elutasíthatjuk vagy elfogadhatjuk, mintha az eredeti algoritmust használnánk. A megmaradt sorozatokat pontos, de ugyanakkor lassabb algoritmusokkal ellenőrizzük. Az algoritmusok ismertetése mellett elemezzük azok futási idejét, az egy sorozat ellenőrzésére jutó átlagos műveletszámok felhasználásával.

A dolgozat másik célja az általunk kidolgozott lineáris futásidejű pontos algoritmus, valamint gyorsított változatának bemutatása.

A dolgozat felépítése a következő: a dolgozat 2. részében ismertetjük Havel és Hakimi algoritmusát és annak egy gyorsított változatát, valamint Erdős és Gallai algoritmusát és annak két gyorsított változatát. Ezek után a 3. részben megadunk néhány közelítő algoritmust és ismertetjük az azok alkalmazásához szükséges műveletigényeket. Röviden ismertetünk a különböző sorozatok számának kiszámítására vonatkozó eredményeket a 4. részben. Az ismertetett algoritmusok hatékonyságát az 5. részben vizsgáljuk meg. Elsőként a pontos algoritmusokat vetjük össze azok műveletigénye alapján, majd megindokoljuk a kapott eredményeinket és áttérünk a közelítő és pontos algoritmusok együttes felhasználására. Az utóbbi esetben is megadjuk az összetett tesztek elvégzéséhez szükséges átlagos műveletszámokat. Az elemzések során tapasztalt megfigyelések alapján megadunk két új algoritmust a 6. részben. Végül a 7. részben összefoglaljuk az elvégzett vizsgálatokat és eredményeinket, valamint a munka folytatására vonatkozó jövőbeli terveinket.

2. Pontos algoritmusok

Mielőtt elkezdenénk a szakirodalomból ismert algoritmusok ismertetését, bevezetünk néhány fontosabb fogalmat. A G gráf i -edik csúcsát jelöljük v_i -vel és v_i foka legyen d_i , a gráfhoz tartozó fokszorozat legyen $F = (f_1, \dots, f_n)$.

Azt mondjuk, hogy az F fokszorozat *szabályos*, ha $n \geq 2$ és f_i , $2 \leq i \leq n$ értékek egész számok és monoton csökkenőek, azaz $f_1 \geq \dots \geq f_n$. Azokat a szabályos fokszorozatokat, amelyekhez megadható egy olyan G egyszerű gráf, amelynek fokszorozata F , *helyreállíthatónak*, *grafikusnak* vagy *megvalósíthatónak* nevezzük. Ezen kívül szükségünk van még a fokszorozat első k tagjának összegére is, amit H_k -val jelölünk.

Az i -edik pontban történő ellenőrzés során a sorozat első i elemét a sorozat *fejének*, az összes azt követő elemet pedig a sorozat *farkának* nevezzük.

2.1. Havel-Hakimi algoritmus (HH)

Az algoritmust elsőként Havel [9] írta le, majd később Hakimi [8] tőle függetlenül publikálta ugyanezt az eredményt, innen adódik a Havel-Hakimi algoritmus elnevezés.

1. TÉTEL. (Havel, Hakimi [8, 9]) Az $F = (f_1, \dots, f_n)$ foksorozat akkor és csak akkor helyreállítható, ha az $F' = (f_2 - 1, f_3 - 1, \dots, f_{f_1} - 1, f_{f_1+1} - 1, f_{f_1+2}, \dots, f_{n-1}, f_n)$ sorozat is helyreállítható.

Bizonyítás. (Lásd [8, 9]). Ha az $F' = (f_2 - 1, f_3 - 1, \dots, f_{f_1} - 1, f_{f_1+1} - 1, f_{f_1+2}, \dots, f_{n-1}, f_n)$ sorozat helyreállítható, akkor a hozzá tartozó G' gráfhoz egy új v_1 csúcsot hozzávéve, majd a $v_2, \dots, v_{f_1+1}$ csúcsokkal összekötve megkapjuk az eredeti $F = (f_1, \dots, f_n)$ foksorozatot. Ezzel megkaptuk az állítás egyik felét.

Az állítás másik feléhez azt kell belátnunk, hogy ha a sorozat helyreállítható, akkor az első v_1 csúcs igényét az öt követő f_1 darab csúcs felhasználásával kielégítve a megmaradt foksorozat is helyreállítható marad. Mivel tudjuk, hogy az eredeti gráf helyreállítható volt, így elegendő a kezdeti gráfból kiindulva sorra elhagyni annak csúcsait a hozzájuk tartozó élekkel együtt. Látható, hogy ezt a módszert követve a gráf tulajdonság nem változik meg, azaz a csúcsokhoz tartozó foksorozat továbbra is helyreállítható marad. $\square$

A tételen alapuló algoritmus nagy hátránya, hogy felhasználása során minden egyes lépésben újra kell rendezni a sorozatot. Ezért az algoritmus futási ideje legrosszabb esetben akkor is $\Omega(n^2)$, ha lineáris futási idejű rendező algoritmust használunk.

A 1. tételben szereplő ellenőrzéseket valósítja meg a következő algoritmus.

Bemenet. n : a foksorozat hosszát adja meg;

F : az ellenőrizendő foksorozat.

Kimenet. L : logikai változó ($L = false$ azt jelenti, hogy a bemenetként megadott F sorozat nem helyreállítható, $L = true$ esetén az F sorozat helyreállítható.)

HAVEL-HAKIMI(n, F, L)

```

01 while  $f_1 > 0$                                 ▷ 01. sor: amíg nem értünk a sorozat végére
02     for  $j = 2$  to  $f_1 + 1$                     ▷ 02–04. sor: kielégítjük az első fok igényét
03          $f_j = f_j - 1$ 
04     end
05      $f_1 = 0$                                     ▷ 05. sor: nullázzuk az első elemet
06      $sort\_descending(F)$                         ▷ 06. sor: újrendezzük a sort
07 end
08  $t = 0$                                             ▷ 08. sor: ideiglenes változó nullázása
09 for  $i = 1$  to  $n$                                 ▷ 09–11. sor: a megmaradt sorozat elemeinek összegzése
10      $t = t + f_i$ 
11 end
12 if  $t == 0$                                       ▷ 12–16. sor: helyreállíthatóság eldöntése
13      $L = true$                                     ▷ 13. sor: helyreállítható
14 else
15      $L = false$                                     ▷ 15. sor: nem helyreállítható
16 end
17 return  $L$                                        ▷ 17. sor: eredmény visszaadása

```

2.2. Eltoló Havel-Hakimi algoritmus (HHE)

Az ellenőrizendő sorozat egyre kisebb és kisebb részének rendezését megvalósíthatjuk úgy is, hogy csupán azokat az elemeket toljuk el a sorozatban, amelyekre nem teljesül a monotonitás. Ezzel a rendezéshez szükséges idő nagy részét megtakaríthatjuk. Ez az algoritmus az eltoló Havel-Hakimi algoritmus.

2.3. Erdős-Gallai algoritmus (EG)

Erdős és Gallai 1960-ban adtak meg olyan ellenőrzési módot, amely a sorozatot nem állítja elő, csupán eldönti, hogy az helyreállítható vagy sem.

2. TÉTEL. (Erdős, Gallai [5]) *A szabályos $F = (f_1, \dots, f_n)$ sorozat akkor és csak akkor helyreállítható, ha*

$$\sum_{i=1}^n f_i \text{ páros} \quad (1)$$

és

$$\sum_{i=1}^j f_i - j(j-1) \leq \sum_{k=j+1}^n \min(j, f_k) \quad (j = 1, \dots, n-1). \quad (2)$$

Bizonyítás. Lásd [3, 5]. A tétel szükségességét könnyű belátni, hiszen a gráfhoz egy élet hozzávéve a fokszámok összege kettővel növekszik, így a fokszámok összegének párosnak kell lennie. A második feltétel bal oldala a fejnek a farok felhasználásával kielégítendő fokszám igényét adja meg, mivel az első j csúcs igényéből levonjuk a fejen belül kielégíthető kapacitást, ami legfeljebb $j(j-1)$. A feltétel jobb oldala pedig a farok által kielégíthető kapacitást adja meg a következő módon: a farok elemei nem tudnak kielégíteni több kapacitást, mint amennyi egy-egy adott csúcs hozzájárulása vagy mint a fej elemszáma.

A tétel szükségességét több lépésben bizonyítjuk. Defináljunk a $v_1, \dots, v_n$ csúcsokból alkotott gráfhoz egy olyan fokszámsorozatot, amelyre teljesül, hogy $d_i \leq f_i$ minden $1 \leq i \leq n$ -re ahol d_i jelöli a v_i csúcs fokszámát. Adott $f_1 \geq \dots \geq f_n$ fokszorozatra, amely teljesíti a megadott feltételeket, minden lépésben készítünk egy részrealizációt és ezekkel a lépésekkel bizonyítjuk a tétel elégségességét.

Induljunk ki egy n csúcsú üres gráfból, továbbá legyen r a legnagyobb olyan érték, amelyre $d_i = f_i$ teljesül minden $1 \leq i < r$ esetén. r -et *kritikus indexnek* nevezzük. Kezdetben $r = 1$, leszámítva azt az esetet, ha minden fokszám 0, mert ebben az esetben készen vagyunk.

Amíg $r \leq n$, addig újabb és újabb lépéseket teszünk úgy, hogy részrealizációkat hozunk létre, amelyekre az $f_r - d_r$ különbség csökken. Közben mindvégig figyelünk rá, hogy a v_i csúcs fokszáma ne változzon, ha $i < r$. Addig folytatjuk a módszert, amíg a kapott részrealizáció realizációja az $f_1 \geq \dots \geq f_n$ fokszorozatnak. Jelöljük $N(v)$ -vel a v csúcs szomszédainak halmazát.

1. eset: $v_r v_i \notin E(G)$ valamely v_i , $d_i < f_i$ esetén. Ekkor $E(G)$ -hez hozzáadjuk a $v_r v_i$ élt.

2. eset: $v_r v_i \notin E(G)$, $i < r$. Mivel $d_i = f_i \geq f_r > d_r$, létezik $u \in N(v_i) - N(v_r)$. Ha $f_r - d_r \geq 2$, akkor helyettesítsük az uv_i élt az uv_r és $v_i v_r$ élekkel. Ha $f_r - d_r = 1$, akkor $\sum f_i - \sum d_i$ paritása miatt létezik $k > r$, hogy $d_k < f_k$. Ekkor az első esethez jutottunk újra, kivéve, ha $v_r v_k \in E(g)$, ilyenkor viszont a $v_r v_k$ és uv_i éleket az uv_r és $v_i v_r$ élekkel helyettesítjük.
3. eset: $v_1, \dots, v_{r-1} \in N(v_r)$ és $d_k \neq \min(r, f_k)$ valamely $k > r$ -re. Tudjuk, hogy $d_k \leq f_k$ és $d_k \leq r$, tehát $d_k < \min(r, f_k)$. Ekkor újra az első esethez értünk, leszámítva azt az esetet, ha $v_k v_r \in E(G)$. Mivel $d_k < r$, ezért létezik $i < r$, hogy $v_k v_i \notin E(G)$. $d_i > d_r$, így létezik $u \in N(v_i) - N(v_r)$. uv_i -t az uv_r és $v_i v_r$ élekkel helyettesítjük.
4. eset: $v_1, \dots, v_{r-1} \in N(v_r)$ és $v_i v_j \notin E(G)$, $i < j < r$. Ekkor a második esethez jutottunk, kivéve, ha $v_i, v_j \in N(v_r)$. $d_i \geq d_j > d_r$, így létezik $u \in N(v_i) - N(v_r)$ és $w \in N(v_j) - N(v_r)$. $u, w \notin N(v_r) \Rightarrow$ ismét a második esethez jutottunk, kivéve, ha $u, w \in \{v_{r+1}, \dots, v_n\}$. Ekkor az uv_i és uv_j éleket a $v_i v_j$ és uv_r élekkel helyettesítjük.

Amennyiben egyik eset sem áll fenn, a $v_1, \dots, v_r$ csúcsok közül bármely kettő között fut él és $d_k = \min(r, f_k)$, ha $k > r$, így

$$\sum_{i=1}^r d_i = r(r-1) + \sum_{k=r+1}^n \min(r, f_k).$$

Mivel a második feltétel teljesül, és a jobb oldal a $\sum_{i=1}^r f_i$ felső becslése, így r -re már teljesül, hogy $d_r = f_r$, tehát r -et növelve folytatható az eljárás. $\square$

A tételen alapuló algoritmus futási ideje a legjobb $\Theta(n)$ és legrosszabb $\Theta(n^2)$ között változik.

A közelmúltban Tripathi és munkatársai [32] publikáltak egy konstruktív bizonyítást, amely helyreállítható sorozatok esetén a helyreállítást is elvégzi. Ennek az algoritmusnak a futási ideje legrosszabb esetben $\Theta(n^3)$.

A 2. tételben leírt feltételeket ellenőrzi a következő algoritmus.

Bemenet. n : a fokszorozat hosszát adja meg;

F : az ellenőrizendő fokszorozat.

Kimenet. L : logikai változó ($L = false$ azt jelenti, hogy a bemenetként megadott F sorozat nem helyreállítható, $L = true$ esetén az F sorozat helyreállítható).

ERDŐS-GALLAI(n, F, L)

```

01  $H_1 = f_1$                                 ▷ 01. sor: első  $H$  érték megadása
02 for  $i = 2$  to  $n$                             ▷ 02–04. sor: további  $H$  értékek számítása
03      $H_i = H_{i-1} + f_i$ 
04 end
05 if  $H_n$  páratlan                            ▷ 05–08. sor: paritás ellenőrzés
06      $L = false$                                 ▷ 06–07. sor: hibás sorozat elutasítása
07 return  $L$ 
```

```

08 end
09 for  $j = 1$  to  $n - 1$  ▷ 09–18. sor: sorozat ellenőrzése
10      $R = 0$  ▷ 10. sor: munkaváltozó nullázása
11     for  $k = j + 1$  to  $n$  ▷ 11–13. sor: a tételben szereplő összeg kiszámítása
12          $R = R + \min(j, f_k)$ 
13     end
14     if  $H_j - j(j - 1) > R$  ▷ 14–17. sor: feltétel ellenőrzése
15          $L = false$  ▷ 15–16. sor: hibás sor elutasítása
16     return  $L$ 
17 end
18 end
19  $L = true$  ▷ 19–20. sor: jó sorozat elfogadása
20 return  $L$ 

```

2.4. Rövidített Erdős-Gallai algoritmus (EGR)

Tripathi et al. 2003-as cikkükben [31] leírnak egy lehetséges rövidítési módot Erdős és Gallai módszerére. A (2) egyenlőtlenséget elegendő csak azokban az esetekben ellenőrizni, ahol a $H_i > i(i - 1)$ feltétel teljesül.

1. LEMMA. (Tripathi et al. [31]) *Egy szabályos $F = (f_1, \dots, f_n)$ foksorozat akkor és csak akkor jó, ha*

$$H_n \text{ páros} \tag{3}$$

és

$$H_i - i(i - 1) \leq \sum_{k=i+1}^n \min(i, f_k), \tag{4}$$

ahol

$$i = 1, \dots, \max_{1 \leq k \leq n} (k \mid k(k - 1) < H_k). \tag{5}$$

Bizonyítás. Lásd [31]. □

A 1. lemmában megadott tesztet valósítja meg a következő algoritmus.

Bemenet. n : a foksorozat hosszát adja meg;

F : az ellenőrizendő foksorozat.

Kimenet. L : logikai változó ($L = false$ azt jelenti, hogy a bemenetként megadott F sorozat nem helyreállítható, $L = true$ esetén az F sorozat helyreállítható).

ERDŐS-GALLAI-SHORTENED(n, F, L)

```

01  $H_1 = f_1$  ▷ 01. sor:  $H_1$  beállítása
02 for  $i = 2$  to  $n$  ▷ 02–04. sor:  $H$  értékek számítása
03      $H_i = H_{i-1} + f_i$ 
04 end
05 if  $H_n$  páratlan ▷ 05–08. sor: paritás ellenőrzés

```

```

06   $L = false$                                 ▷ 06–07. sor: rossz sorozat elutasítása
07  return  $L$ 
08 end
09  $l = 1$                                         ▷ 09. sor: kezdeti érték beállítása
10 while  $l < n$  and  $l(l-1) < H_l$                 ▷ 10–20. sor:  $l$  sorozat tesztelése
11      $R = 0$                                     ▷ 14. sor: munkaváltozó nullázása
12     for  $k = j + 1$  to  $n$                     ▷ 12–14. sor: a tétel szummájának számítása
13          $R = R + \min(j, f_k)$ 
14     end
15     if  $H_j - j(j-1) > R_j$                     ▷ 15–18. sor: feltétel ellenőrzése
16          $L = false$                             ▷ 16–17. sor: rossz sorozat elutasítása
17         return  $L$ 
18     end
19      $l = l + 1$                                 ▷ 19. sor: ciklusváltozó növelése
20 end
21  $L = true$                                     ▷ 21–22. sor: jó sorozat elfogadása
22 return  $L$ 

```

2.5. Ugró Erdős-Gallai algoritmus (EGU)

Tripathi és Vijay [31] megadtak egy további lehetséges rövidítési módot is. A rövidítés azt használja ki, hogy ha egy ellenőrizendő sorozatban egy fokszám többször is előfordul, akkor elegendő azt csupán egyszer ellenőrizni, még hozzá abban a pontban, ahol teljesül, hogy $f_i \neq f_{i+1}$. A sorozatok utolsó elemeit pedig nem szükséges ellenőrizni.

Vezessük be a g_i jelölést azokra az i indexekre, amelyekre teljesülnek az $f_i \neq f_{i+1}$ és $i \neq n$ feltételek. Nevezzük ezeket a pontokat *ellenőrző pontoknak*. Ezzel a jelöléssel tehát a sorozat $f_{g_1}, \dots, f_{g_q}$ elemei ellenőrző elemek és q jelöli a sorozat ellenőrző elemeinek számát.

3. TÉTEL. (Tripathy, Vijay [31]) *A szabályos $F = (f_1, \dots, f_n)$ sorozat akkor és csak akkor helyreállítható, ha igaz az, hogy*

$$H_n \text{ páros} \tag{6}$$

és

$$H_{g_i} - g_i(g_i - 1) \leq \sum_{k=g_i+1}^{n-1} \min(i, f_k) \quad (i = 1, \dots, q). \tag{7}$$

Bizonyítás. Lásd [31]. □

A 3. tételben leírt feltételeket ellenőrzi a következő algoritmus.

Bemenet. n : a fokszorozat hosszát adja meg;

F : az ellenőrizendő fokszorozat.

Kimenet. L : logikai változó ($L = false$ azt jelenti, hogy a bemenetként megadott F sorozat nem helyreállítható, $L = true$ esetén az F sorozat helyreállítható).

ERDŐS-GALLAI-JUMPING(n, F, L)

```

01  $H_1 = f_1$                                 ▷ 01. sor:  $H_1$  megadása
02 for  $i = 2$  to  $n$                             ▷ 02–04. sor:  $H$  értékek számítása
03      $H_i = H_{i-1} + f_i$ 
04 end
05 if  $H_n$  páratlan                            ▷ 05–08. sor: paritás ellenőrzés
06      $L = false$                                 ▷ 06–07. sor: hibás sorozat elutasítása
07 return  $L$ 
08 end
09 for  $j = 1$  to  $n - 1$                         ▷ 09–23. sor: sorozat ellenőrzése
10     if  $f_j = f_{j+1}$         ▷ 10–12. sor: eldöntjük, hogy az adott pont ellenőrzőpont-e
11         continue
12     end
13      $R = 0$                                 ▷ 13. sor: munkaváltozó nullázása
14     for  $k = j + 1$  to  $n$                     ▷ 14–16. sor: a tétel szummájának számítása
15          $R = R + \min(j, f_k)$ 
16     end
17     if  $H_j - j(j - 1) > R$                 ▷ 17. sor: a tétel feltételének ellenőrzése
18          $L = false$                             ▷ 18–19. sor: rossz sorozat elutasítása
19         return  $L$ 
20     end
21 end
22  $L = true$                                 ▷ 22–23. sor: jó sorozat elfogadása
23 return  $L$ 

```

3. Közelítő algoritmusok

Ebben a részben ismertetünk néhány közelítő algoritmust, amelyek alkalmasak lehetnek a pontos algoritmusok előtt futtatva előtesztelni a sorozatokat, azzal a céllal, hogy nagy részüket gyorsan elutasítsák, majd a megmaradt sorozatokat pontos algoritmusok segítségével ellenőrizzük.

Az alábbi algoritmusok bemutatása során röviden jellemezzük is az egyes algoritmusok műveletigényét. Ezek a műveletigények az algoritmusok önmagukban való használata esetén értendőek, azaz ha csak az adott algoritmust használjuk ellenőrzésre. Itt azonban megjegyezzük, hogy a gyakorlatban ezeket a közelítő algoritmusokat egymás után alkalmazzuk az el nem utasított sorozatokon. Mivel a műveletigényük jelentős része közös, így egyes tesztek egymás után való alkalmazása során a műveletigény nem az adott teszt teljes műveletigényével növekszik, hanem annak csak egy kis hányadával.

3.1. Gyors teszt (FT)

Első tesztünk egy roppant gyors teszt, melynek alapja a következő: amennyiben egy sorozat $p > 0$ nem nulla értéket tartalmaz, úgy a sorozat első eleme nem lehet nagyobb, mint $p - 1$, hiszen nagyobb igényt a sorozat többi tagja nem képes kielégíteni.

2. LEMMA. *Ha F egy helyreállítható sorozat és $p > 0$ az F nem nulla elemeinek száma, akkor*

$$H_1 \leq p - 1.$$

Bizonyítás. Ha az F sorozat nem tartalmaz legalább $H_1 + 1$ nemnulla elemet, akkor biztosan nem helyreállítható. $\square$

A teszt elvégzésének műveletigénye pontosan 1 feltételvizsgálat, mivel az ellenőrizendő sorozatokról feltettük, hogy azok monoton csökkenő sorrendben rendezettek, így elegendő a sorozat f_{f_1} tagjáról eldönteni, hogy nulla vagy sem. Szimulációs eredményeink alapján tudjuk, hogy a rossz sorozatoknak megközelítőleg harmada kiszűrhető ezzel a teszttel.

A 2. lemmában megadott feltételt ellenőrzi a következő algoritmus.

Bemenet. F : az ellenőrizendő fokszorozat.

Kimenet. L : logikai változó ($L = false$ azt jelenti, hogy a bemenetként megadott F sorozat nem helyreállítható, $L = true$ esetén az F sorozatról nem tudja eldönteni a teszt, hogy helyreállítható-e vagy sem).

FAST-TEST(F, L)

```

01 if  $F_{f_1+1} == 0$                                 ▷ 01–04. sor: feltétel ellenőrzése
02    $L = false$                                        ▷ 02–03. sor: rossz sorozat elutasítása
03   return  $L$ 
04 end
05  $L = true$                                            ▷ 05–06. sor: nem elutasított sorozat elfogadása
06 return  $L$ 

```

3.2. Paritás teszt (PT)

Második tesztünk az Erdős-Gallai algoritmus (1) feltételét foglalja magába. Mivel a teszt (2) részében ellenőrizendő feltétel ellenőrzéséhez mindenképpen ki kellene számolnunk a H_n értéket, így gyakorlatilag nem jelent plusz munkát ennek a tesztnek az elvégzése. Ez a teszt nagyon hasznos, tekintve, hogy megközelítőleg minden második szabályos sorozat páratlan, így ezeket gyorsan elutasíthatjuk.

3. LEMMA. *Ha $n \geq 1$ és az F sorozat helyreállítható, akkor H_n páros.*

Bizonyítás. Egyszerű gráfokban minden él kettővel növeli a foksámok összegét. $\square$

A teszt elvégzésének műveletigénye pontosan $n + 1$, azaz n összeadás és egy feltételvizsgálat (az értékadásokat nem számítjuk).

A 3. lemmában leírt feltételt ellenőrzi a következő algoritmus.

Bemenet. n : a fokszorozat hosszát adja meg;

F : az ellenőrizendő fokszorozat.

Kimenet. L : logikai változó ($L = false$ azt jelenti, hogy a bemenetként megadott F sorozat nem helyreállítható, $L = true$ esetén az F sorozatról nem tudja eldönteni a teszt, hogy helyreállítható-e vagy sem).

PARITY-TEST(n, F, L)

```

01  $H_1 = f_1$                                 ▷ 01. sor:  $H_1$  érték beállítása
02 for  $i = 2$  to  $n$                             ▷ 02–04. sor:  $H$  értékek számítása
03      $H_i = H_{i-1} + f_i$ 
04 end
05 if  $H_n$  páratlan                            ▷ 05–08. sor: paritás ellenőrzés
06      $L = false$                                 ▷ 06–07. sor: hibás sorozat elutasítása
07     return  $L$ 
08 end
09  $L = true$                                     ▷ 09–10. sor: páros paritású sor elfogadása
10 return  $L$ 

```

3.3. Binomiális teszt (BT)

Harmadik tesztünkben a fej belső éleinek számát binomiális együtthatóval közelítjük.

4. LEMMA. Ha $n \geq 1$ és F helyreállítható sorozat, akkor

$$H_i \leq H_n - H_i + i(i - 1). \quad (8)$$

Bizonyítás. A (8) egyenlőtlenség azt fejezi ki, hogy az első f_i csúcs fokszám szükségletét $i(i - 1)$ belső és $H_n - H_i$ külső kapacitás felhasználásával próbáljuk kielégíteni. $\square$

A teszt elvégzésének maximális műveletigénye $n + 5$, azaz $n + 1$ összeadás, 2 kivonás, egy szorzás és egy feltételvizsgálat.

A 4. lemmában leírt feltételeket ellenőrzi a következő algoritmus.

Bemenet. n : a fokszorozat hosszát adja meg;

F : az ellenőrizendő fokszorozat.

Kimenet. L : logikai változó ($L = false$ azt jelenti, hogy a bemenetként megadott F sorozat nem helyreállítható, $L = true$ esetén az F sorozatról nem tudja eldönteni a teszt, hogy helyreállítható-e vagy sem).

BINOMIAL-TEST(n, F, L)

```

01  $H_1 = f_1$                                 ▷ 01. sor:  $H_1$  számítása
02 for  $i = 2$  to  $n$                             ▷ 02–04. sor:  $H$  értékek számítása

```

```

03    $H_i = H_{i-1} + f_i$ 
04 end
05 for  $j = 1$  to  $n - 1$                                 ▷ 05–10. sor: sorozat ellenőrzése
06   if  $H_j - j(j - 1) > H_n - H_j$                     ▷ 06–09. sor: lemma feltételének ellenőrzése
07      $L = false$                                        ▷ 07–08. sor: rossz sorozat elutasítása
08   return  $L$ 
09 end
10 end
11  $L = true$                                            ▷ 11–12. sor: jó sorozat elfogadása
12 return  $L$ 

```

3.4. Binomiális teszt ugrással (BTJ)

Negyedik tesztünknel bevezetjük az ugró Erdős-Gallai algoritmus esetében már alkalmazott ellenőrző pontokat, ezzel csökkentve az elvégzendő tesztek számát. Mivel a binomiális teszt csak közelítő algoritmus, így a tesztek csökkentése egyúttal a pontosság csökkenését is eredményezi, azonban a teszt romlása jóval kisebb mértékű, mint az ellenőrző pontok alkalmazásával elért gyorsulás.

4. Leszámlálási eredmények

Ebben a részben a gráfsorozatok számára, valamint az n -szabályos és n -páros sorozatok számának arányára vonatkozóan ismertetünk néhány formulát.

Az általunk vizsgált sorozat számára vonatkozóan Sloane és Ploffe [29], valamint Stanley [30] könyveiben és az *On-Line Encyclopedia of Integer Sequences* [27] honlapon találhatunk adatokat.

Könnyen belátható [17], hogy az n -szabályos sorozatok R_n száma

$$R_n = \binom{2n-1}{n}. \quad (9)$$

1987-ben Ascher adott meg explicit formulát az n -hosszú páros sorozatok számának meghatározására.

5. LEMMA. (Ascher [1], Sloane, Ploffe [29]) *Ha $n \geq 1$, akkor az n -szabályos páros sorozatok száma*

$$E_n = \frac{1}{2} \left(\binom{2n-1}{n} + \binom{n-1}{\lfloor n/2 \rfloor} \right). \quad (10)$$

Bizonyítás. Lásd [1]. □

Az n -szabályos sorozatok növekedésének nagyságrendjét adja meg a következő lemma.

6. LEMMA. *Ha $n \geq 1$, akkor*

$$\frac{R_{n+2}}{R_{n+1}} > \frac{R_{n+1}}{R_n} \quad (11)$$

$$\lim_{n \rightarrow \infty} \frac{R_{n+1}}{R_n} = 4, \quad (12)$$

továbbá

$$\frac{4^n}{\sqrt{4\pi n}} \left(1 - \frac{1}{2n}\right) < R_n < \frac{4^n}{\sqrt{4\pi n}} \left(1 - \frac{1}{8n+8}\right). \quad (13)$$

Bizonyítás. Lásd [13]. □

A következő lemma megadja az E_n értékek növekedésének nagyságrendjét.

7. LEMMA. *Ha $n \geq 1$, akkor*

$$\frac{E_{n+2}}{E_{n+1}} > \frac{E_{n+1}}{E_n} \quad (14)$$

és

$$\lim_{n \rightarrow \infty} \frac{E_{n+1}}{E_n} = 4, \quad (15)$$

továbbá

$$\frac{4^n}{\sqrt{\pi n}} (1 - D_3(n)) < E_n < \frac{4^n}{\sqrt{\pi n}} (1 - D_4(n)), \quad (16)$$

ahol a $D_3(n)$ és $D_4(n)$ monoton csökkenő és nullához tart.

Bizonyítás. Lásd [13]. □

Ahogy az 1. táblázatból is leolvasható, az E_n/R_n arány monoton csökken és $1/2$ -hez tart.

1. KÖVETKEZMÉNY. *Ha $n \geq 1$, akkor*

$$\frac{E_{n+1}}{R_{n+1}} < \frac{E_n}{R_n} \quad (17)$$

és

$$\lim_{n \rightarrow \infty} \frac{E_n}{R_n} = \frac{1}{2}. \quad (18)$$

Bizonyítás. Lásd [13]. □

Az ugrópontok száma nagymértékben befolyásolja az azokat felhasználó algoritmus futási idejét.

Egy F n -korlátos fokszorozat különböző elemeinek számát a sorozat szivárványszámának nevezzük és r_n -nel jelöljük.

8. LEMMA. *Legyen F egy véletlen n -szabályos fokszorozat és r_n annak a szivárványszáma. Várható értéke a következő alakban adható meg:*

$$E(r_n) = \frac{n}{2} + \frac{n}{4n-2} = \frac{n}{2} + O(1). \quad (19)$$

n	R_n	E_n	E_n/R_n
1	1	1	1,0000000000
2	3	2	0,6666666667
3	10	6	0,6000000000
4	35	19	0,5428571429
5	126	66	0,5238095238
6	462	236	0,5108225108
7	1 716	868	0,5058275058
8	6 435	3 235	0,5027195027
9	24 310	12 190	0,5014397367
10	92 378	46 252	0,5006819806
11	352 716	176 484	0,5003572279
12	1 352 078	676 270	0,5001708481
13	5 200 300	2 600 612	0,5000888410
14	20 058 300	10 030 008	0,5000427753
15	77 558 760	38 781 096	0,5000221252
16	300 540 195	150 273 315	0,5000107057
17	1 166 803 110	583 407 990	0,5000055151
18	4 537 567 650	2 268 795 980	0,5000026787
19	17 672 631 900	8 836 340 260	0,5000013756
20	68 923 264 410	34 461 678 394	0,5000006702
21	269 128 937 220	134 564 560 988	0,5000003432
22	1 052 049 481 860	526 024 917 288	0,5000001676
23	4 116 715 363 800	2 058 358 034 616	0,5000000857
24	16 123 801 841 550	8 061 901 596 814	0,5000000419
25	63 205 303 218 876	31 602 652 961 516	0,5000000214
26	247 959 266 474 052	123 979 635 837 176	0,5000000105
27	973 469 712 824 056	486 734 861 612 328	0,5000000053
28	3 824 345 300 380 220	1 912 172 660 219 260	0,5000000026
29	15 033 633 249 770 520	7 516 816 644 943 559	0,5000000013
30	59 132 290 782 430 712	29 566 145 429 994 736	0,5000000007

1. táblázat. A szabályos és páros sorozatok száma és azok aránya.

Bizonyítás. Lásd [13]. □

A következő lemma megadja a nullmentes sorozatok arányát a szabályos sorozatokhoz viszonyítva.

9. LEMMA. *Az n -szabályos nullmentes sorozatok száma*

$$R_n^z = \binom{2n-2}{n-1} \quad (20)$$

és

$$\lim_{n \rightarrow \infty} \frac{R_n^z}{R_n} = \frac{1}{2}. \quad (21)$$

Bizonyítás. Lásd [13]. □

10. LEMMA. *Ha $n > 1$, akkor az n -csúcsú gráfok száma megadható az $n-1$ csúcsú megvalósítható fokszorozatok és az n hosszú jó nullmentes sorozatok számának összegeként, azaz*

$$G_n = Z_n + G_{n-1}.$$

A G értékek nagyságrendjére vonatkozóan Burns adott meg aszimptotikus korlátot.

11. LEMMA. (Burns [2]) *Léteznek olyan c és C pozitív konstansok, hogy a G_n értékre az alábbi korlát adható:*

$$\frac{4^n}{cn} < G_n < \frac{4^n}{(\log n)^C \sqrt{n}}. \quad (22)$$

Bizonyítás. Lásd [2]. □

Ebből következik, hogy a grafikus sorozatok számának sűrűsége a páros sorozatok között a nullához tart.

2. KÖVETKEZMÉNY. *Ha $n \geq 1$, akkor létezik olyan pozitív C konstans, hogy*

$$\frac{G_n}{E_n} < \frac{1}{(\log_2 n)^C} \quad (23)$$

és

$$\lim_{n \rightarrow \infty} \frac{G_n}{E_n} = 0. \quad (24)$$

Bizonyítás. (23) a (10) és a (22) közvetlen következménye, a (23) egyenlőtlenségből pedig következik a (24) határérték. □

További eredmények és összefüggések találhatóak a [13] cikkben. Szimulációs eredményeink szerint például az is igaz, hogy a páros sorozatoknak körülbelül fele nullmentes.

5. Elemzések

Az algoritmusok elemzése során összevetjük az ismert pontos algoritmusok elvégzéséhez szükséges műveletigényeket, illetve megvizsgáljuk, hogy az Erdős-Gallai algoritmus különböző változatainak mennyi ellenőrzésre van szüksége a rossz sorozatok elutasításához és a jó sorozatok elfogadásához. Az utóbbi értékek fontossága kézenfekvő, hiszen minél kevesebb művelettel meg tudjuk állapítani egy sorozatról, hogy rossz vagy jó, annál több sorozatot ellenőrizhetünk le adott idő alatt.

Közelítő algoritmusaink elemzése során megvizsgáljuk a tesztek elvégzéséhez szükséges műveletigényeket, valamint azt is, hogy a közelítő algoritmusokat a pontos algoritmusok előtt alkalmazva hogyan változik a futási idő. Kitérünk arra is, hogy az egyes közelítő algoritmusok mennyi sorozatot fogadnak el és hogy mennyi az algoritmusok által elfogadott sorozatok és az összes vizsgált sorozat számának aránya.

5.1. Pontos algoritmusok

A korábbiakban ismertetett algoritmusokat összevetettük műveletigényeik szerint. Ezen azt értjük, hogy megvalósítottuk az algoritmusokat C nyelven, majd megszámoltuk (számlálót építve a programba), hogy bizonyos n értékek esetén egy-egy adott algoritmus hány műveletet végez. A sorozatok előállításának idejét nem számítottuk be, az értékadásokat azonban ugyanúgy műveletként értelmeztük, mint például az összeadást vagy a kivonást. A bonyolultabb műveleteket, mint például a szorzás vagy az osztás, nem számítottuk nagyobb súllyal, mint a többit. A számítások során csak szabályos sorozatokat használtunk fel, azaz minden F sorozat monoton csökkenő volt. A műveletigényekkel kapcsolatos eredményeinket a 2. táblázat tartalmazza.

A táblázatból leolvasható, hogy a legnagyobb műveletigényű algoritmus az eredeti HH algoritmus, itt azonban nem szabad elfeledkeznünk róla, hogy ez a műveletigény részben azért ilyen nagy, mert minden lépésben újra kell rendeznünk az ellenőrzés alatt álló – iterációnként változó – sorozatot. Ha azonban a teljes sorozat rendezése helyett csak eltoljuk a rossz helyre került elemeket, akkor az így kapott HHE algoritmus műveletigénye már közel a felére csökken és így az EG algoritmus műveletigényének duplájához közelít.

Az ellenőrző pontok bevezetésével az Erdős-Gallai algoritmus műveletigénye közel a felére csökken, továbbá az tapasztalható, hogy a kapott algoritmus és a korábbi algoritmusok műveletigényének aránya egyre kisebb, azaz az algoritmus lépésről lépésre gyorsul.

Az EG és az EGU algoritmusok közti sebesség-különbség szemléltetésére megvizsgáltuk, hogy a két algoritmusnak hány iterációra van szüksége a hibás sorozatok elutasításához. Ezek az adatok láthatóak a 3. táblázatban az EG, a 4. táblázatban pedig az EGU algoritmusra vonatkozóan. Az elutasítások pontos értékei helyett az összes elutasítással vett hányadosok láthatóak a táblázatban, mivel

n	HH	HHE	EG	EGU
2	13	14	40	4
3	23	34	27	12
4	33	50	30	16
5	47	64	32	21
6	67	79	39	26
7	92	95	48	32
8	122	111	58	37
9	149	130	69	43
10	188	149	80	49
11	229	170	93	56
12	285	193	106	63
13	335	216	120	70
14	402	241	135	77
15	472	267	151	85

2. táblázat. Pontos algoritmusok egy sorozat ellenőrzéséhez szükséges átlagos műveletszáma.

$n \setminus i$	1	2	3	4	5	6	7	8	9	10
2	1,00									
3	0,75	0,25								
4	0,71	0,23	0,06							
5	0,63	0,26	0,09	0,01						
6	0,61	0,24	0,11	0,03	0,004					
7	0,58	0,24	0,12	0,05	0,012	0,001				
8	0,56	0,23	0,12	0,06	0,022	0,004	0,0003			
9	0,55	0,22	0,12	0,07	0,032	0,010	0,0014	0,0001		
10	0,54	0,22	0,12	0,07	0,039	0,016	0,0040	0,0005	0,00002	
11	0,53	0,21	0,11	0,07	0,044	0,022	0,0079	0,0016	0,00015	0,000004

3. táblázat. Adott pontban az EG teszt által elutasított sorok számának aránya az összes elutasított sorozathoz viszonyítva.

ezekről jobban leolvasható az adott iterációs lépésben elutasított sorozatok aránya. Megjegyezzük, hogy a táblázat nem tartalmaz a jó sorozatokra vonatkozó értékeket, hiszen jó sorozatok esetén nem beszélhetünk elutasításról – éppen ezért a jó sorozatok ellenőrzése a legköltségesebb abban az értelemben, hogy egy jó sorozat esetében az összes lehetséges iterációt el kell végeznünk (kivéve, ha például alkalmazzuk az 1. lemmát vagy más, a jó sorozatokat gyorsan felismerő módszert).

A 3. táblázatból leolvasható, hogy négyzetes műveletigénye ellenére az Erdős-Gallai algoritmus nagyon hatékonyan dolgozik abban az értelemben, hogy a sorozatok nagy részét gyorsan elutasítja.

Ahogy az a 4. táblázatból leolvasható, az ugró Erdős-Gallai algoritmus még ennél is hatékonyabb. Az első ellenőrzés során a sorozatoknak ugyanakkora hányadát tudja elutasítani, mint az eredeti, azonban a legnagyobb iterációs lépés száma – rossz sorozat esetén – $\lceil n/2 \rceil$, azaz a szükséges ellenőrző lépések száma összefügg az n érték

$n \backslash i$	1	2	3	4	5	6
2	1,000					
3	1,000					
4	0,943	0,057				
5	0,910	0,090				
6	0,873	0,124	0,004			
7	0,845	0,143	0,012			
8	0,820	0,157	0,022	0,001		
9	0,801	0,164	0,034	0,002		
10	0,784	0,167	0,044	0,005	0,0001	
11	0,770	0,168	0,052	0,009	0,0005	
12	0,758	0,168	0,059	0,014	0,0013	0,00002

4. táblázat. Adott ellenőrzőpontban az EGU algoritmus által elutasított sorok számának aránya az összes elutasított sorozathoz viszonyítva.

paritásával: ha n páratlan, akkor az ellenőrzési iterációk maximális száma $(n-1)/2$, ha pedig páros, akkor ez a szám $n/2$.

Az eddig ismertetett eredményeket összegezve tehát azt mondhatjuk, hogy a Havel-Hakimi algoritmus a rossz helyre került értékek eltolásával felgyorsítható. Az Erdős-Gallai algoritmust kiegészítve az ellenőrző pontokkal az ugró algoritmus az n értékének növelésével lépésről lépésre gyorsul. $n = 15$ esetben például az EGU algoritmushoz szükséges műveletszámhoz képest műveletigénye annak csak hat tizede.

5.2. Közelítő algoritmusok

Ebben a részben a 3. részben ismertetett közelítő algoritmusokra vonatkozó adatokat mutatunk be. A közelítő algoritmusok vizsgálata során azt vizsgáltuk, hogy egy adott algoritmus összesen hány sorozatot fogad el és hogy ezeknek a sorozatoknak az elfogadásához milyen műveletigény tartozik.

Az algoritmusok vizsgálata során arra törekszünk, hogy a pontos, négyzetes idejű algoritmusokat felgyorsítsuk néhány gyors közelítő algoritmus felhasználásával. A korábban ismertetett közelítő algoritmusok által elfogadott sorozatok száma látható a 5. táblázatban. Megjegyezzük, hogy ezek a számok úgy értendők, hogy a különböző algoritmusokat sorban, egymásra épülve végezzük, tehát a binomiális teszt egyben magába foglalja a gyors és a paritás teszteket is. Ezt fejezik ki a felső indexekben látható c -k. Ez alól kivételt képeznek a binomiális és az ugró binomiális tesztek, mivel ezek gyakorlatilag ugyanannak a tesztnek különböző variánsai.

Ahogy az a táblázatból is jól látható, a gyors teszt a rossz sorozatoknak körülbelül egyharmadát képes elutasítani konstans időben, ez nagyon fontos szerepet játszhat a későbbiekben, például a HH algoritmus kiegészítéseként. A paritás teszt a megmaradt sorozatok közül kiszűri a páratlan paritású, azaz biztosan rossz sorozatokat. A tévesen elfogadott sorozatok aránya ebben az esetben nagyobb,

n	R_n	FT_n^c	PT_n^c	BT_n^c	BTJ_n^c	G_n
1	1	1	1	1	1	1
2	3	2	2	2	2	2
3	10	6	4	4	4	4
4	35	21	13	11	11	11
5	126	77	41	31	31	31
6	462	287	151	103	103	102
7	1 716	1 079	547	349	349	342
8	6 435	4 082	2 066	1 256	1 260	1 213
9	24 310	15 522	7 786	4 577	4 604	4 361
10	92 378	59 280	29 728	17 040	17 178	16 016
11	352 716	227 240	113 708	63 944	64 544	59 348
12	1 352 078	873 886	437 262	242 218	244 672	222 117
13	5 200 300	3 370 030	1 685 334	922 369	931 987	836 315
14	20 058 300	13 027 730	6 515 042	3 530 534	3 567 561	3 166 852
15	77 558 760	50 469 890	25 236 122	13 563 764	13 704 763	12 042 620

5. táblázat. Az összes sorozat, gyors, paritás, binomiális, ugró binomiális – sorban alkalmazott – tesztek által elfogadott és jó sorozatok száma az n érték függvényében.

viszont az ellenőrzés elvégzésének műveletigénye lineáris, míg a gyors teszt konstans időben végzi el ezt a munkát. A binomiális teszt az első két teszten megfelelt sorozatok megközelítőleg feléről el tudja dönteni, hogy azok hibásak-e, így ez a teszt is nagyon hatékony, műveletigénye a paritás teszt részeredményeinek felhasználásával már nagyon alacsony. A binomiális teszt ellenőrző pontokkal kiegészített változata már kevésbé pontos, azonban műveletigénye alacsonyabb. A közelítő algoritmusokkal kapcsolatban mért műveletigényeket mutatja a 6. táblázat. Az EG tesztre vonatkozó adataink a korábbi táblázatoktól eltérőek lehetnek, mivel a különböző méréseinket más-más programokkal és más környezetekben (MATLAB, C) valósítottuk meg.

A 6. táblázatban látható műveletigényekből leolvasható, hogy az ugró Erdős-Gallai teszt esetében is megéri a közelítő algoritmusokat előtesztként alkalmazni. Az ismertetett algoritmusaink közül a leggyorsabb a konstans műveletigényű gyors teszt. A gyors teszttel kiegészített ugró Erdős-Gallai algoritmusra vonatkozóan a 7. táblázatban láthatóak egy sorozatra jutó átlagos műveletszámok. Méréseink alapján azt mondhatjuk, hogy az n érték növelésével a gyors teszt hatékonysága nem romlik, éppen ezért - valamint konstans műveletigénye miatt - érdemes beépítenünk a pontos algoritmusainkba is. Szimulációs eredményeink alapján arra a következtetésre jutottunk, hogy az Erdős-Gallai teszt és annak gyorsított változata is abban az esetben gyorsítható a legnagyobb mértékben közelítő algoritmusokkal, ha a gyors és a paritás teszteket alkalmazzuk. Megjegyezzük, hogy a paritás teszt egyébként is része az algoritmusoknak.

Ahogy azt már láthattuk, az Erdős-Gallai algoritmus és ugró változata nagymértékben gyorsítható közelítő algoritmus felhasználásával, azaz a dolgozat első részében kitűzött célunkat részben már elértük. Térjünk vissza a Havel-Hakimi

n	$\frac{O_{FT_n}}{R_n}$	$\frac{O_{PT_n}}{R_n}$	$\frac{O_{BT_m}}{R_n}$	$\frac{O_{EGU_n}}{R_n}$	$\frac{O_{BTJ_n}}{R_n}$	$\frac{O_{EGU_n}}{R_n}$	$\frac{O_{EGU_n}}{R_n}$
2	1	1	2	2	2	2	2
3	2	3	7	9	5	7	8
4	2	5	11	14	7	11	11
5	2	6	13	17	9	14	15
6	2	6	15	20	11	18	18
7	2	7	17	23	12	21	22
8	2	8	19	26	26	25	25
9	2	8	20	29	15	28	29
10	2	9	22	33	17	32	33
11	2	10	24	37	18	36	37
12	2	10	26	40	20	40	41
13	2	11	28	44	21	45	45
14	2	12	29	48	23	49	50

6. táblázat. A gyors teszt, paritás ellenőrzés, binomiális, EGU, ugró binomiális és EGU tesztek átlagos műveletigényei egy sorozatra nézve (egészre kerekítve).

n	2	3	4	5	6	7	8	9	10	11	12	13	14
$\frac{O_{EGU_n}}{R_n}$	2	8	11	15	18	22	25	29	33	37	41	45	50
$\frac{O_{EGUF_n}}{R_n}$	2	6	9	12	14	17	20	23	26	30	33	37	41

7. táblázat. Az ugró Erdős-Gallai és a gyors tesztrel kiegészített ugró Erdős-Gallai algoritmus átlagos műveletszámai egy sorozat esetén.

n	HH	HHGy	HHE	HHEGy
2	3	3	4	4
3	10	7	13	9
4	17	13	22	16
5	25	17	31	20
6	35	24	40	26
7	48	30	50	31
8	63	38	62	37
9	77	45	74	43
10	96	55	88	49
11	117	65	102	56
12	143	78	118	64
13	168	90	135	72
14	199	105	152	80

8. táblázat. HH, HHGy, HHE és HHEGy algoritmusok felhasználása során egy sorozat ellenőrzésére szükséges átlagos lépésszámok.

algoritmushoz. Ennél az algoritmusnál gyors tesztünk komolyabb hatásfokkal képes dolgozni, mint korábban hiszen a HH minden lépésben újrendezi a sorozatot, így minden egyes lépésben elvégezhető a teszt, ami így összesen n plusz műveletet eredményez, de az n érték növekedésével egyre nagyobb gyorsulást eredményez, mivel a jó sorozatok relatív száma a nullához tart [13]. A Havel-Hakimi algoritmus és a gyors teszt együttes használatáról (HHGy) a 8. táblázatban láthatóak mérési adatok.

A táblázatban szereplő értékek alapján elmondhatjuk, hogy a gyors tesztrel kiegészített Havel-Hakimi algoritmus gyorsabb, mint az eredeti Erdős-Gallai módszer és kicsivel elmarad az ugró Erdős-Gallai algoritmustól.

További közelítő algoritmusok és eredmények találhatóak a [13] cikkben.

6. Javasolt új algoritmusok

Az előző részben megvizsgált közelítő és pontos algoritmusok vizsgálata során tett megfigyeléseink alapján megadunk egy-egy új, javított módszert a Havel-Hakimi és Erdős-Gallai algoritmusok felgyorsítására.

6.1. Havel-Hakimi-Eltoló algoritmus gyors tesztrel (HHEGy)

A 3.1. részben ismertetett közelítő algoritmussal kapcsolatos vizsgálataink során azt tapasztaltuk, hogy azt beépítve a Havel-Hakimi algoritmusba az algoritmus műveletigénye megközelítőleg a felére csökken (elegendően nagy n esetén) és az eredeti Erdős-Gallai algoritmusnál gyorsabb működésre képes.

4. TÉTEL. Az $F = (f_1, \dots, f_n)$ fokszorozat akkor és csak akkor helyreállítható, ha $f_{f_1} > 0$ és a $F' = (f_2 - 1, f_3 - 1, \dots, f_{f_1} - 1, f_{f_1+1} - 1, f_{f_1+2}, \dots, f_{n-1}, f_n)$ sorozat

is helyreállítható.

Bizonyítás. Tételünk az eredeti algoritmusnak azt a hiányosságát küszöböli ki, hogy nem veszi figyelembe, hogy egy adott fokszámigény kielégítése lehetséges-e az adott körülmények között. Mivel azonban minden iterációban újrendezzük a sorozatot, így könnyedén ellenőrizhető, hogy érdemes-e még folytatnunk az ellenőrzést, vagy a sorozatunkról már kiderült, hogy nem helyreállítható. $\square$

Az új algoritmussal kapcsolatos mérési eredményeket tartalmaz a 8. táblázat.

Az 4. tétel feltételeit ellenőrzi a következő algoritmus.

Bemenet. n : a fokszorozat hosszát adja meg;

F : az ellenőrizendő fokszorozat.

Kimenet. L : logikai változó ($L = false$ azt jelenti, hogy a bemenetként megadott F sorozat nem helyreállítható, $L = true$ esetén az F sorozat helyreállítható.)

Munkaváltozók. e : az aktuálisan kielégítendő csúcs indexe;

u : az utolsó csúcs indexe, amit összekötünk az e -edik csúccsal.

HAVEL-HAKIMI-FAST-SHIFTING(n, F, L)

```

01  $e = 1$                                 ▷ 01. sor: a legnagyobb fokszám indexének beállítása
02 while  $f_e > 0$                           ▷ 02–32. sor: sorozat ellenőrzése
03      $u = f_e + e$  ▷ 03. sor: az utolsó elem ami része az aktuális igény kielégítésének
04     if  $f_u == 0$                           ▷ 04–07. sor: gyors teszt
05          $L = false$                         ▷ 05–06. sor: rossz sorozat elutasítása
06         return
07     end
08     if  $f_u == f_{u+1}$                       ▷ 08–29. sor: ellenőrzése 'eltolással'
09          $i = e + 1$                         ▷ 09. sor: első egyforma elem index
10         if  $f_u \neq f_i$                     ▷ 10–14. sor: első egyforma elem keresése
11             while  $f_i \neq f_u$ 
12                  $i = i + 1$ 
13             end
14         end
15         for  $j = e + 1$  to  $i$                 ▷ 15–17. sor: eltolás mentes rész csökkentése
16              $f_j = f_j - 1$ 
17         end
18          $j = u + 2$                         ▷ 18. sor:  $j$  beállítása
19         while  $f_u = f_j$                   ▷ 19–21 sor: utolsó ugyanolyan elem megkeresése
20              $j = j + 1$ 
21         end
22         for  $k = 0$  to  $u - i + 1$  ▷ 22–25. sor: fokszámok csökkentése hátulról haladva
23              $j = j - 1$ 
24              $f_j = f_j - 1$ 
25         end
26          $f_e = 0$                           ▷ 26. sor: a kielégített első elem nullázása
27          $e = e + 1$                         ▷ 27. sor: új első elem beállítása
28     else                                ▷ 28–34. sor: ellenőrzés 'eltolás' nélkül

```

```

29      for  $j = e + 1$  to  $last + 1$   ▷ 28–30. sor: fejelem igényének kielégítése
30           $f_j = f_j - 1$ 
31      end
32       $f_e = 0$   ▷ 32. sor: a kielégített első elem nullázása
33       $e = e + 1$   ▷ 33. sor: új első elem beállítása
34  end
35 end
36 if  $f_n == 0$   ▷ 36. sor: ha az utolsó igényt is sikeresen kielégítettük
37      $L = true$   ▷ 37. sor: elfogadó állapot beállítása
38 else
39      $L = false$   ▷ 39. sor: elutasító állapot beállítása
40 end
41 return  $L$   ▷ 41. sor: visszatérés az eredménnyel

```

6.2. Lineáris-Erdős-Gallai-Ugró algoritmus (EGLU)

Közelítő algoritmusaink elemzése során azt figyeltük meg, hogy az Erdős-Gallai ugró algoritmust tovább gyorsíthatjuk, ha kihasználjuk az f sorozat monotonitását, tehát, hogy a feltételben szereplő összegzést nem kell minden egyes iterációban elvégezni, mivel a szummában szereplő min függvény egy w_i pontig (súlypont) mindig az első, majd attól kezdve mindig a második paraméterét fogja visszaadni. A monotonitás miatt pedig ez az érték csak az egyik irányban változhat, így elég az egyes iterációk között néhány lépésben eltolni. Ezáltal nyerünk egy explicit képletet a szummára, amit korábban mindig újra és újra kiszámítottunk.

5. TÉTEL. *A szabályos $F = (f_1, \dots, f_n)$ sorozat akkor és csak akkor helyreállítható, ha igaz az, hogy*

$$H_n \text{ páros} \quad (25)$$

és

$$H_{g_i} \leq \begin{cases} H_n - H_{g_i} + g_i(g_i - 1), & \text{ha } w_i \leq g_i \\ H_n - H_{w_i} + g_i(w_i - 1), & \text{ha } w_i > g_i. \end{cases} \quad (26)$$

Bizonyítás. Lásd [13]. Az ellenőrzőpontok elégségességét Tripathi és munkatársai [31] már bebizonyították. A tételben megadott feltétel ezeket az ellenőrzéseket végzi el, kihasználva a sorozat elemeinek monoton csökkenését, azaz a

$$\sum_{k=g_i+1}^n \min(i, f_k) \quad (27)$$

összeget nem számolja újra minden esetben, pontosabban nem ebben a formában végzi el a számítást, hanem explicit módon.

A kifejezés értéke a (28) formában adható meg, mégpedig azért, mert a sorozat monotonitása garantálja, hogy a w_i súlypont előtt a $\min(i, f_k)$ kifejezés értéke minden esetben i , majd a w_i -nél nagyobb k -k esetében mindvégig f_k . Ebből következik, hogy

n	2	3	4	5	6	7	8	9	10	11	12	13	14	15
EGU	4	12	16	21	26	32	37	43	49	56	63	70	77	85
$\frac{\text{EGU}}{n}$	2.0	4.0	4.0	4.2	4.3	4.6	4.6	4.8	4.9	5.1	5.3	5.4	5.5	5.7
EGLU	12	15	17	19	21	23	25	27	29	31	33	35	37	39
$\frac{\text{EGLU}}{n}$	6.0	5.0	4.3	3.8	3.5	3.3	3.1	3.0	2.9	2.8	2.8	2.7	2.6	2.6

9. táblázat. Az ugró és az Erdős-Gallai lineáris ugró algoritmusok egy sorozatra jutó átlagos műveletigénye.

$$\sum_{k=g_i+1}^{n-1} \min(i, f_n) = \begin{cases} H_n - H_{g_i}, & \text{ha } w_i \leq g_i \\ H_n - H_{w_i} + g_i(w_i - g_i), & \text{ha } w_i > g_i. \end{cases} \quad (28)$$

Az eddigiek alapján az eredeti feltételt átírhatjuk a következő alakba:

$$H_{g_i} - g_i(g_i - 1) \leq \begin{cases} H_n - H_{g_i}, & \text{ha } w_i \leq g_i \\ H_n - H_{w_i} + g_i(w_i - g_i), & \text{ha } w_i > g_i. \end{cases} \quad (29)$$

A (29) egyenlőtlenséget átrendezve megkapjuk a

$$H_{g_i} \leq \begin{cases} H_n - H_{g_i} + g_i(g_i - 1), & \text{ha } w_i \leq g_i \\ H_n - H_{w_i} + g_i(w_i - 1), & \text{ha } w_i > g_i \end{cases} \quad (30)$$

kifejezést, ami pontosan egyezik a (26) egyenlőtlenséggel. $\square$

A most megadott tétel alapján megvalósított algoritmus és az eddigi legjobb (ugró Erdős-Gallai) algoritmus sorozatonkénti átlagos műveletszámait, valamint a sorozat egyetlen elemére jutó átlagos műveletszámot tartalmazza a 9. táblázat. A táblázatban szereplő értékek számítása során csak nullmentes sorozatokat ellenőriztünk, ezért az itt szereplő értékek eltérhetnek a korábbi táblázatokban szereplőktől.

A táblázatból leolvasható, hogy az átlagos műveletszám a lineáris algoritmus esetében kevesebb, mint fele annyi, mint az ugró algoritmus esetében és az n érték növelésével minden lépésben ugyanannyival növekszik. Az utóbbi azért fontos, mert így az n növelésével lépésről lépésre nagyobb az új algoritmussal elért gyorsulás a korábbiakhoz képest. Az utóbbi kijelentés azonban nem meglepő, ha figyelembe vesszük, hogy a korábbi ismert algoritmusok négyzetesek, míg az új algoritmus lineáris futási idejű. Jól látható, hogy az régi módszer esetén a sorozatok egy eleméhez tartozó átlagos műveletszám az n érték növekvésével együtt nőtt, az új módszernél azonban ez a szám lépésről lépésre csökken.

Az 5. tétel feltételeit ellenőrzi a következő algoritmus.

Bemenet. n : a fokszorozat hosszát adja meg;

F : az ellenőrizendő fokszorozat.

Kimenet. L : logikai változó ($L = false$ azt jelenti, hogy a bemenetként megadott F sorozat nem helyreállítható, $L = true$ esetén az F sorozat helyreállítható.)

ERDŐS-GALLAI-LINEAR-JUMPING(n, F, L)

```

01  $H_1 = f_1$                                 ▷ 01. sor:  $H_1$  számítása
02 for  $i = 2$  to  $n$                             ▷ 02–04. sor:  $H$  értékek számítása
03      $H_i = H_{i-1} + f_i$ 
04 end
05 if  $H_n$  páratlan                            ▷ 05–08. sor: paritás ellenőrzés
06      $L = false$                                 ▷ 06–07. sor: hibás sorozat elutasítása
07 return
08 end
09  $w = n$                                     ▷ 09. sor: súlypont inicializálása
10 for  $i = 1$  to  $n - 1$                             ▷ 10–28. sor: sorozat ellenőrzése
11     if  $f_i == f_{i+1}$                             ▷ 11–13 sor: ellenőrzőpont tulajdonság ellenőrzése
12         continue                                ▷ 12. sor: nem ellenőrzőpont átlépése
13     end
14     while  $w > 1$  and  $f_w \leq i$                 ▷ 14–15. sor: súlypont frissítése
15          $w = w - 1$ 
16     end
17     if  $w < i$                                 ▷ 17–21. sor: súlypont az ellenőrzőpont előtt van
18         if  $H_i > H_n - H_i + i(i - 1)$         ▷ 18–21. sor: tétel feltételének ellenőrzése
19              $L = false$                             ▷ 19–20. sor: rossz sorozat elutasítása
20             return
21         end
22     else                                ▷ 22–27. sor: súlypont az ellenőrzőpont után van
23         if  $H_i > H_n - H_w + i(w - 1)$     ▷ 23–26. sor: tétel feltételének ellenőrzése
24              $L = false$                             ▷ 24–25. sor: rossz sorozat elutasítása
25             return
26         end
27     end
28 end
29  $L = true$                                     ▷ 29–30. sor: jó sorozat elfogadása
30 return  $L$ 

```

6. TÉTEL. *A lineáris Erdős-Gallai ugró algoritmus műveletigénye lineáris.*

Bizonyítás. Az 1. sor $O(1)$, a 2–4. sorok $\Theta(n)$, az 5–8., valamint a 9. sor egyaránt $O(1)$, a 10–28. sorok $O(n)$, a 29–30. sorok elvégzéséhez pedig $O(1)$ idő szükséges. Az algoritmus teljes műveletigénye tehát $\Theta(n)$. $\square$

7. Összefoglalás

Dolgozatunk első részében rövid áttekintést adtunk a felvetett problémáról, annak alkalmazási területeiről és az ismert megoldási módszerekről.

Az áttekintést követően ismertettük a két legfontosabb pontos algoritmust és azok néhány változatát.

A dolgozat harmadik részében megadtunk néhány új közelítő algoritmust, azzal a céllal, hogy ezeket előtesztekként használva felgyorsítsuk a második részben ismertetett pontos algoritmusokat, valamint röviden jellemeztük ezeket a közelítő algoritmusokat műveletigényeik szerint.

A dolgozat negyedik része leszámplálási eredményeket tartalmaz az összes és páros sorozatok számára vonatkozóan, valamint ismertetjük a nullákat is tartalmazó, és a nullmentes sorozatok száma közti összefüggést.

A dolgozat ötödik részében elemeztük az ismert pontos algoritmusokat műveletigényük szerint és rövid magyarázatot adtunk a köztük tapasztalható eltérések okára. A pontos algoritmusok elemzését követően pontos és közelítő algoritmusok együttes alkalmazásait vizsgáltuk, ahol azt a következtetést vontuk le, hogy az ismertetett tesztek felhasználásával a klasszikus algoritmusok (és az ugró Erdős-Gallai algoritmus is) gyorsítható, azaz megtakarítható a fokszorozatok ellenőrzéséhez szükséges műveletigény egy része, és kevesebb mint felére csökkenthető a Havel-Hakimi algoritmus műveletigénye. Ezzel lényegében a dolgozat elején kitűzött célunkat sikeresen elértük. További közelítő algoritmusok és eredmények találhatóak még a [13, 15] cikkekben.

Dolgozatunk hatodik részében az elemzések eredményei alapján megadtunk két új algoritmust a Havel-Hakimi és Erdős-Gallai gyorsításaiként. Az egyik ilyen algoritmus a gyorsított Havel-Hakimi algoritmus, amelynek műveletigénye már alacsonyabb, mint az Erdős-Gallai algoritmusé. A másik új algoritmus a lineáris Erdős-Gallai módszer, amellyel az eredeti négyzetes futási idejű módszert lineárisá alakítottuk. A lineáris Erdős-Gallai algoritmussal részletesen foglalkozunk a [12, 15] cikkekben. A lineáris Erdős-Gallai algoritmus minden szóba jövő sorozat ellenőrzésére alkalmas változatával és annak hálózati szerver-kliens megvalósításával foglalkozik a [23] dolgozat és a [14] cikk, amelyek a grafikus fokszorozatok számát, azaz a G_n értékeit $n = 29$ -ig tartalmazzák. Az általunk kiszámolt új értékeket 2011. november 15. óta tartalmazza az OEIS is [28].

Az eddigieket összefoglalva tehát ismertettünk két klasszikus algoritmust, majd bemutattunk néhány közelítő algoritmust, megvizsgáltuk azok együttes alkalmazásának előnyeit és hátrányait, majd a tapasztalt eredmények alapján megadtunk két új algoritmust, amelyek az eredeti algoritmusoknál gyorsabb munkára képesek. Ezek közül az egyik az eredeti négyzetes algoritmus munkáját lineáris időben látja el.

A jövőben szeretnénk az újonnan megadott algoritmusokat páros gráfokra is kiterjeszteni, valamint felhasználni fokszorozatok vizsgálatához [7, 24].

A dolgozatban felhasznált szimulációs programok és részletes eredmények letölthetőek a szerzők <http://people.inf.elte.hu/lulsaai/Holzhacker> című oldaláról.

Hivatkozások

- [1] ASCHER, M.: *Mu torere: an analysis of a Maori game*. Math. Mag. **60**, (1987) 90–100. $\Rightarrow$ 13

- [2] BURNS, J. M.: *The Number of Degree Sequences of Graphs*. PhD Dissertation, MIT, 2007. $\Rightarrow$ 16
- [3] CHOUDUM, S. A.: *A simple proof of the Erdős-Gallai theorem on graph sequences*. Bull. Austral. Math. Soc. **33**, (1986) 67–70. $\Rightarrow$ 6
- [4] CHUNGPHAISAN, V.: *Conditions for sequences to be r -graphical*. Discrete Math. **7**, (1974) 31–39. $\Rightarrow$ 3
- [5] ERDŐS, P., GALLAI, T.: *Gráfok előírt fokú pontokkal*. Mat. Lapok **11**, (1960) 264–274. $\Rightarrow$ 3, 6
- [6] ERDŐS, P. L., MIKLÓS, I., TOROCZKAI, Z.: *A simple Havel-Hakimi type algorithm to realize graphical degree sequences of directed graphs*. Elec. J. Comb. **17**, (2010) R66. $\Rightarrow$ 3
- [7] FRANK, A.: *Connections in Combinatorial Optimization*. Calderon Oxford Univ. Press, Oxford, (2011). $\Rightarrow$ 27
- [8] HAKIMI, S. L.: *On the realizability of a set of integers as degrees of the vertices of a simple graph*. J. SIAM Appl. Math. **10**, (1962) 496–506. $\Rightarrow$ 3, 4, 5
- [9] HAVEL, V.: *A remark on the existence of finite graphs (Czech)*. Časopis Pěst. Mat. **80**, (1955) 477–480. $\Rightarrow$ 3, 4, 5
- [10] IVÁNYI, A.: *Reconstruction of complete interval tournaments*. Acta Univ. Sapientiae, Inform., **1**, (2009) 71–88. $\Rightarrow$ 3
- [11] IVÁNYI, A.: *Reconstruction of complete interval tournaments. II*. Acta Univ. Sapientiae, Math., **2**, (2010) 47–71. $\Rightarrow$ 3
- [12] IVÁNYI, A., LUCZ, L.: *Erdős-Gallai test in linear time*. Comb., benyújtva. $\Rightarrow$ 27
- [13] IVÁNYI, A., LUCZ, L., MÓRI, F. T., SÓTÉR, P.: *On the Erdős-Gallai and Havel-Hakimi algorithms*. Acta Univ. Sapientiae, Inform., **3**, (2011) 230–268. Elérhető: <http://www.acta.sapientia.ro/acta-info/> $\Rightarrow$ 3, 14, 16, 22, 24, 27
- [14] IVÁNYI, A., LUCZ, L., PIRZADA, S.: *Parallel Erdős-Gallai algorithm*. Centr. Eur. J. Oper. Res., benyújtva. $\Rightarrow$ 27
- [15] IVÁNYI, A., LUCZ, L., MÓRI, F. T.: *Lineáris Erdős-Gallai teszt*. Alk. Mat. Lapok, benyújtva. $\Rightarrow$ 27
- [16] IVÁNYI, A., PIRZADA, S.: *Comparison based ranking*. In (ed. A. Iványi): Algorithms of Informatics, **3**, AnTonCom, Budapest, (2011), 1209–1256. $\Rightarrow$ 3
- [17] JÁRAI, A. (ed.): *Introduction to Mathematics (Hungarian)*. ELTE Eötvös Kiadó, Budapest, (2005). $\Rightarrow$ 13

- [18] KIM, H., TOROCZKAI, Z., MIKLÓS, I., ERDŐS, P. L., SZÉKELY, L. A.: *Degree-based graph construction*. J. Physics: Math. Theor. **42**, (2009) 392–401. $\Rightarrow 3$
- [19] KLEITMAN, D. J., WANG, D. L.: *Algorithms for constructing graphs and digraphs with given valences and factors*. Disc. Mat., **6**, (1973) 79–88. $\Rightarrow 3$
- [20] LAMAR, M. D.: *Algorithms for realizing degree sequences of directed graphs*. arXiv (2009). Elérhető: <http://arxiv.org/abs/0906.0343> $\Rightarrow 3$
- [21] LANDAU, H. G.: *On dominance relations and the structure of animal societies. III. The condition for a score sequence*. Bull. Math. Biophys. **15**, (1953) 143–148. $\Rightarrow 3$
- [22] LILJEROS, F., EDLING, C. R., AMARAL, L., STANLEY, H., ÁBERG, Y.: *The web of human sexual contacts*. Nature, **411**, (2001) 907–908. $\Rightarrow 3$
- [23] LUCZ, L. (témavezető: IVÁNYI, A.): *Párhuzamos Erdős-Gallai algoritmus*. TDK dolgozat, ELTE IK, Budapest, 2011. $\Rightarrow 3, 27$
- [24] LUCZ, L., IVÁNYI, A.: *Testing and enumeration of football sequences*. In (ed. Z. Csörnyei): Abstracts of MaCS 2012, Siófok, Február 9–12, 2012. $\Rightarrow 27$
- [25] NEWMAN, M. E. J., BARABÁSI, A. L.: *The Structure and Dynamics of Networks*. Princeton University Press, Princeton, (2006). $\Rightarrow 3$
- [26] ÖZKAN, S.: *Generalization of the Erdős-Gallai inequality*. Ars Combin. **98** (2011) 295–302. $\Rightarrow 3$
- [27] SLOANE N. J. A.: *The On-line Encyclopedia of Integer Sequences*. Elérhető: <http://oeis.org/>. $\Rightarrow 3, 13$
- [28] SLOANE N. J. A.: *Number of graphical partitions*. In (ed. N. J. A. Sloane): The On-line Encyclopedia of Integer Sequences (2011). Elérhető: <http://oeis.org/A004251>. $\Rightarrow 3, 27$
- [29] SLOANE, N. J. A., PLOUFFE, S.: *The Encyclopedia of Integer Sequences*., Academic Press, (1995). $\Rightarrow 13$
- [30] STANLEY, R. P.: *Enumerative Combinatorics*, Cambridge University Press, Cambridge, (1997). $\Rightarrow 13$
- [31] TRIPATHI, A., VIJAY, S.: *A note on a theorem of Erdős & Gallai*. Disc. Math. **265**, (2003) 417–420. $\Rightarrow 3, 8, 9, 24$
- [32] TRIPATHI, A., VENUGOPALAN, S., WEST, D. B.: *A short constructive proof of the Erdős-Gallai characterization of graphic lists*. Disc. Math. **310**, (2010) 833–834. $\Rightarrow 7$