

TDK dolgozat

Lucz Loránd



EÖTVÖS LORÁND TUDOMÁNYEGYETEM
INFORMATIKAI KAR
KOMPUTERALGEBRA TANSZÉK

Párhuzamos Erdős-Gallai algoritmus

TDK dolgozat

Témavezető:
Dr. Iványi Antal Miklós
egyetemi tanár

Készítette:
Lucz Loránd
II. évfolyam
Programtervező informatikus MSc

Budapest, 2011

Tartalomjegyzék

1. Bevezetés	3
2. Algoritmusok	4
2.1. Havel-Hakimi algoritmus (HH)	4
2.2. Erdős-Gallai algoritmus (EG)	5
2.3. Rövidített Erdős-Gallai algoritmus (EGR)	5
2.4. Ugró Erdős-Gallai algoritmus (EGU)	6
2.5. Lineáris Ugró Erdős-Gallai algoritmus (EGLU)	6
3. Leszámlálási eredmények	7
4. A Holzhacker program	10
4.1. Kiszolgáló program	12
4.2. Feladatokat kiosztó algoritmus (DJ)	15
4.3. Kliens program	16
4.4. Felsoroló Erdős-Gallai algoritmus (EGE)	18
5. A feladat részekre bontása	22
5.1. Naív megközelítés	22
5.2. Felosztás korábbi n értékekre alapozva	22
5.3. Felosztás a műveletszámok alapján	23
5.4. Felosztás a páros sorozatok száma alapján	23
6. Eredmények	26
7. Összefoglalás	26
Hivatkozások	29

1. Bevezetés

A bennünket körülvevő világot sokféleképpen modellezhetjük. Az egyik lehetséges módszer a világ gráfokkal történő reprezentálása. A körülöttünk levő objektumokat logikailag összekapcsolhatjuk, majd ezeket a kapcsolatokat ábrázolhatjuk gráfok segítségével. Egyszerű példaként tekinthetünk például egy családfát, ahol a gráf egy csúcsa jelöl egy személyt, a gráf élei pedig a szülő-gyermek viszonyokat. Ez alapján mindenki a szüleivel és gyermekeivel áll közvetlen kapcsolatban. A gráfok csúcsai minden esetben legfeljebb egyszeresek lehetnek és a kapott gráf hurokél mentes. Az ilyen gráfokat *egyszerű gráfnak* nevezzük.

Mivel manapság már szinte mindent számítógépekkel végzünk, ezért fontos kérdés ezeknek a gráfoknak a tárolása. Ennek egyik lehetséges módja a csúcsok fokszámainak tárolása. Egy gráfból könnyedén felírhatunk egy foksorozatot, megfordítva azonban már nem ilyen egyszerű a probléma megoldása, hiszen nem szükségszerűen tartozik mint sorozathoz gráf és adott sorozathoz több gráf is tartozhat. Ezzel a kérdéssel sok helyen találkozhatunk. Landau [20] az állatok között fennálló dominanciát modellezte ilyen módon, Hakimi [8] kémiai, Iványi és munkatársai pedig sportbeli [10, 11, 13, 15] alkalmazásokra hivatkoztak.

A problémára elsőként Václav Havel javasolt megoldást 1955-ben [9], majd tőle függetlenül 1962-ben Louis Hakimi [8] is publikálta a módszert. Ezért a módszert rendszerint Havel-Hakimi algoritmusként emlegetik. A módszernek bizonyos szempontból előnye, másfelől pedig hátránya, hogy a sorozatokat nem csupán megvizsgálja, hanem a hozzájuk tartozó gráfot is előállítja, ezért a legrosszabb futási ideje nem lehet kisebb, mint négyzetes.

Erdős és Gallai 1960-ban [5] javasolt másik módszert a kérdés eldöntésére, amely már nem állítja elő a gráfot, csupán ellenőriz. Az ő módszerükhöz később Tripathi és Vijay [27] megadott két javaslatot, amellyel az eredeti módszer gyorsítható, de még ezeket felhasználva is nagyon komoly számítási igényrel rendelkezik a feladat.

Az eredeti algoritmusok ugyan csak egyszerű gráfokra alkalmazhatóak, azonban a későbbiekben kiterjesztették őket egyszerű irányított gráfokra [6, 18, 19], valamint irányítatlan [4, 23] és irányított [10, 11] multigráfokra is.

2011-ben megadtuk [13, 21] az Erdős-Gallai módszer egy olyan változatát (EGL), amellyel lineáris időben elvégezhető a vizsgálat. A módszer egy változatát a későbbiekben ismertetjük.

Sloan interneten elérhető *The On-Line Encyclopedia of Integer Sequences* [24] nevű adatbázisában találhatóak adatok arra vonatkozóan, hogy n csúcsú gráfok esetén pontosan hány különböző foksorozat létezik. 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 későbbiekben ismertetett program felhasználásával megadtuk a G_n értékét egészen $n = 29$ -ig. Eredményeink megtalálhatóak az OEIS adatbázis A004251-es sorozatában [24].

Az EGL algoritmust továbbfejlesztve, a minden szóba jövő sorozat ellenőrzésére készített EGP algoritmussal nagyobb n értékekre is meghatároztuk a G_n

értékeit. Ezeket az értékeket egy – több számítógépen párhuzamosan futtatott – hálózati program segítségével számítottuk ki, mivel a számítások futási ideje az n érték növelésével minden lépésben körülbelül négyszeresére nő [13]. Ez az exponenciális növekedés tette szükségessé az algoritmus számítógépes megvalósításának párhuzamosítását.

Dolgozatunk célja a sorozatok gyors ellenőrzési módszereinek ismertetése, valamint a vizsgálatot elvégző program bemutatása.

A dolgozat felépítése a következő: a 2. részben ismertetjük a Havel-Hakimi és az Erdős-Gallai algoritmusokat, leírjuk Tripathi és Vijay gyorsításait, valamint a lineáris Erdős-Gallai ugró algoritmust. A különböző feltételeknek megfelelő sorozatok számával kapcsolatos leszámhlási eredményeket tartalmazza a 3. rész. A számításokat elvégző programot a 4. részben ismertetjük, majd a 5. részben írjuk le a probléma részekre bontásának technikáit, végül az 6. részben összefoglaljuk a leírt módszerekkel és a programmal elért eredményeinket.

2. Algoritmusok

A dolgozatban szereplő algoritmusok forráskódjai megtalálhatóak a <http://people.inf.elte.hu/lulsaai/Holzhacker> honlapon, pszeudokódjai pedig a [13] cikkben és a [21] TDK dolgozatban.

Mielőtt bevezetnénk a felhasznált algoritmusokat, definiáljuk a használt fogalmakat és megadjuk, hogy milyen alapvető feltételezésekkel élünk ezek kapcsán.

A G gráf i -edik csúcsa legyen V_i , V_i foka legyen f_i és G foksorozata legyen $F = (f_1, \dots, f_n)$.

Egy F foksorozat *szabályos*, ha $n \geq 1$ és az f_i ($1 \leq i \leq n$) értékek egész számok, továbbá $n - 1 \geq f_1 \geq \dots \geq f_n \geq 0$. Egy szabályos foksorozatot *megvalósíthatónak* vagy *helyreállíthatónak* nevezünk, ha létezik olyan G egyszerű gráf, amelynek foksorozata F . Egy F szabályos sorozat *páros*, ha elemeinek összege páros.

Szükségünk lesz továbbá a sorozat első k elemének összegére, amit $H_k = \sum_{i=1}^k f_i$ -vel fogunk jelölni.

2.1. Havel-Hakimi algoritmus (HH)

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

1. TÉTEL. (Havel, Hakimi [8, 9]) *Az $F = (f_1, \dots, f_n)$ ($n \geq 3$) szabályos 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]. □

Az algoritmust később kiterjesztették egyszerű irányított gráfokra [6, 18, 19], valamint irányítatlan [4, 23] és irányított [10, 11] multigráfokra is.

A tétel felhasználásával felírható algoritmus nagy hátránya, hogy felhasználása során minden egyes lépésben újra kell rendezni a sorozatot. Másrészt az algoritmus egyesével csökkenti a fokszámokat, ezért például az $F = (n-1, \dots, n-1)$ sorozatnál $\Theta(n^2)$ csökkentést végez. Ezért az algoritmus futási ideje legrosszabb esetben akkor is $\Omega(n^2)$, ha lineáris futási idejű rendező algoritmust használunk.

2.2. 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-e 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, 21]. □

A tétel felhasználásával megvalósítható algoritmus futási ideje a legjobb $\Theta(n)$ és legrosszabb $\Theta(n^2)$ között változik.

A közelmúltban Tripathi és munkatársai [28] publikáltak egy konstruktív bizonyítást, amely helyreállítható sorozatok esetén a helyreállítást is elvégzi. Ennek az algoritmusnak a futási ideje legrosszabb esetben $\Theta(n^3)$.

2.3. Rövidített Erdős-Gallai algoritmus (EGR)

Tripathi és Vijay 2003-as cikkükben [27] leírtak 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. [27]) *Egy szabályos $F = (f_1, \dots, f_n)$ foksorozat akkor és csak akkor helyreállítható, ha*

$$H_n \text{ páros} \quad (3)$$

és

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

ahol

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

Bizonyítás. Lásd [27]. □

2.4. Ugró Erdős-Gallai algoritmus (EGU)

Tripathi és Vijay [27] megadtak egy további lehetséges rövidítési módot is. A rövidítés azt használja ki, hogy ha egy ellenőrizendő sorozatban egy fokszám többször is előfordul, akkor azt elegendő csupán egyszer ellenőrizni, méghozzá 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 az $i \neq n$ feltételek. Ezeket a pontokat *ellenőrzőpontoknak* nevezzük. Ha q jelöli az ellenőrzőpontok számát, akkor az F sorozat $f_{g_1}, \dots, f_{g_q}$ elemei ellenőrzőpontok.

3. TÉTEL. (Tripathi, Vijay [27]) *A szabályos $F = (f_1, \dots, f_n)$ sorozat akkor és csak akkor helyreállítható, ha*

$$H_n \text{ páros} \quad (6)$$

és

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

Bizonyítás. Lásd [27]. □

2.5. Lineáris Ugró Erdős-Gallai algoritmus (EGLU)

Az algoritmust tovább gyorsíthatjuk, ha kihasználjuk az F sorozat monotonitását, tehát azt, hogy a feltételben szereplő összegzést nem kell minden egyes iterációban elvégezni, mivel a szummában szereplő min függvény egy w_i indexig (*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, így nyerünk egy explicit képletet a szummára. A szummák értékének kiszámítása így a legrosszabb esetben is csak lineáris időt vesz igénybe.

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

$$H_n \text{ páros} \quad (8)$$

és

$$H_{g_i} \leq \begin{cases} H_n - H_{g_i} + g_i(g_i - 1), & \text{ha } w_i \leq g_i \\ H_n - H_{w_i} + g_i(w_i - 1), & \text{ha } w_i > g_i. \end{cases} \quad (9)$$

Bizonyítás. Lásd [21].

Az ellenőrzőpontok elégségességét Tripathi és munkatársai [27] már bebizonyították. A tételben megadott feltétel ezeket az ellenőrzéseket végzi el, kihasználva a sorozat elemeinek monoton csökkenését, azaz a

$$\sum_{k=g_i+1}^n \min(i, f_k) \quad (10)$$

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

A kifejezés értéke a (11) formában adható meg, mégpedig azért, mert a sorozat monotonitása garantálja, hogy a w_i súlypontig 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

$$\sum_{k=g_i+1}^n \min(i, f_k) = \begin{cases} H_n - H_{g_i}, & \text{ha } w_i \leq g_i \\ H_n - H_{w_i} + g_i(w_i - g_i), & \text{ha } w_i > g_i. \end{cases} \quad (11)$$

Az eddigiek alapján az eredeti feltételt átírhatjuk a következő alakba:

$$H_{g_i} - g_i(g_i - 1) \leq \begin{cases} H_n - H_{g_i}, & \text{ha } w_i \leq g_i \\ H_n - H_{w_i} + g_i(w_i - g_i), & \text{ha } w_i > g_i. \end{cases} \quad (12)$$

A (12) egyenlőtlenséget átrendezve megkapjuk a

$$H_{g_i} \leq \begin{cases} H_n - H_{g_i} + g_i(g_i - 1), & \text{ha } w_i \leq g_i \\ H_n - H_{w_i} + g_i(w_i - 1), & \text{ha } w_i > g_i \end{cases} \quad (13)$$

kifejezést, ami pontosan egyezik a (9) egyenlőtlenséggel. $\square$

5. TÉTEL. *A lineáris Erdős-Gallai ugró algoritmus futási ideje lineáris.*

Bizonyítás. Lásd [21]. $\square$

3. Leszámlálási eredmények

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

Megmutatható, hogy ha l, u és m egészek, valamint $u \geq l$, $m \geq 1$ és $l \leq f_i \leq u$ ($i = 1, \dots, m$), akkor az (l, u, m) -szabályos sorozatok száma

$$K(l, u, m) = (u - l + 1)^m. \quad (14)$$

Tudjuk, hogy (lásd [17], 65. oldal) ha l, u, m egészek és $u \geq l$, $m \geq 1$, $u \geq f_1 \geq \dots \geq f_n \geq l$, akkor az (l, u, m) -korlátos szabályos sorozatok száma

$$Q(l, u, m) = \binom{u - l + m}{m}. \quad (15)$$

Az összes általunk vizsgált sorozat számára vonatkozóan Sloane és Ploffe [25], valamint Stanley [26] könyveiben és a *The On-Line Encyclopedia of Integer Sequences* [24] honlapon találhatunk adatokat (lásd 1. táblázat).

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

2. LEMMA. (Ascher [1], Sloane, Pfoffe [25]) *Ha $n \geq 1$, akkor a páros sorozatok száma*

$$E_n = \frac{1}{2} \left(\binom{2n-1}{n} + \binom{n-1}{n} \right). \quad (16)$$

Bizonyítás. Lásd [1]. □

A nagy mennyiségű sorozatot ellenőrző program megvalósítása során hasznos lesz a későbbiekben az R_n nagyságrendjére, valamint az R_n és E_n értékek arányára vonatkozó lemma.

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

$$\frac{R_{n+2}}{R_{n+1}} > \frac{R_{n+1}}{R_n}, \quad (17)$$

$$\lim_{n \rightarrow \infty} \frac{R_{n+1}}{R_n} = 4, \quad (18)$$

továbbá

$$\frac{4^n}{\sqrt{4\pi n}} \left(1 - \frac{1}{2n} \right) < R_n < \frac{4^n}{\sqrt{4\pi n}} \left(1 - \frac{1}{8n+8} \right) \quad (19)$$

Bizonyítás. Lásd [13]. □

A következő lemma megadja az E_n értékek nagyságrendjét.

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

$$\frac{E_{n+2}}{E_{n+1}} > \frac{E_{n+1}}{E_n} \quad (20)$$

$$\lim_{n \rightarrow \infty} \frac{E_{n+1}}{E_n} = 4 \quad (21)$$

továbbá

$$\frac{4^n}{\sqrt{\pi n}} (1 - D_3(n)) < E_n < \frac{4^n}{\sqrt{\pi n}} (1 - D_4(n)) \quad (22)$$

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

Bizonyítás. Lásd [13]. □

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

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

$$\frac{E_{n+1}}{R_{n+1}} < \frac{E_n}{R_n} \quad (23)$$

és

$$\lim_{n \rightarrow \infty} \frac{E_n}{R_n} = \frac{1}{2}. \quad (24)$$

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.

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

$$R_n^z = \binom{2n-2}{n-1} \quad (25)$$

és

$$\lim_{n \rightarrow \infty} \frac{R_n^z}{R_n} = \frac{1}{2}. \quad (26)$$

Bizonyítás. Lásd [13]. □

Szimulációs eredményeink szerint [13] az is igaz, hogy a páros sorozatoknak körülbelül a fele nullmentes.

A G értékek nagyságrendjére vonatkozóan Burns adott meg aszimptotikus korlátokat.

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

$$\frac{4^n}{cn} < G_n < \frac{4^n}{(\log n)^C \sqrt{n}}. \quad (27)$$

Bizonyítás. Lásd [2]. □

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

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

$$\frac{G_n}{E_n} < \frac{1}{(\log_2 n)^C} \quad (28)$$

és

$$\lim_{n \rightarrow \infty} \frac{G_n}{E_n} = 0. \quad (29)$$

Bizonyítás. (28) a (16) és a (27) közvetlen következménye és a (28) egyenlőtlenségből következik a (29) határérték. □

További eredmények és összefüggések találhatóak a [13] cikkben.

4. A Holzhacker program

A G_n értékek meghatározása komoly erőforrásigénnyel bír, ezért számításainkat az alábbi technikák felhasználásával igyekeztünk felgyorsítani:

- a feladat részekre bontása és szétosztása több számítógép között,

- a nyilvánvalóan nem grafikus sorozatok elhagyása,
- annak kihasználása, hogy a lexikografikus sorrendben az új sorozatok a részfeladaton belül $O(1)$ várható idő alatt előállíthatóak,
- csak nullmentes sorozatok vizsgálata,
- ellenőrzés elvégzése csak ugrópontokban,
- annak kihasználása, hogy az ellenőrzőpontok és súlypontok listája a részfeladaton belül $O(1)$ várható idő alatt frissíthető, ha a vizsgálandó nullmentes, páros sorozatokat lexikografikus sorrendben állítjuk elő.

A feladat párhuzamosítása a processzorok számával arányosan csökkenti a szükséges futási időt, azaz minél több számítógéppel dolgozunk, annál kevesebb ideig tartanak a számítások.

Nyilvánvalóan nem helyreállítható sorozatoknak tekintjük azokat a sorozatokat, amelyekben a foksorozatok összege páratlan.

A nullás sorozatok vizsgálatának elhagyása pedig azért hasznos, mert a páros sorozatok megközelítőleg felében szerepel nulla. Mivel minden egyszerű gráfhoz tartozó foksorozatra teljesül, hogy egy nullát hozzávéve a kapott sorozathoz továbbra is megadható hozzá egy gráf, ezért ezeket a sorozatokat felesleges megvizsgálnunk, mert G_n számítása során újra leellenőriznénk minden G_{n-1} számítása során korábban már megvizsgált sorozatot. Ezt az ötletet használja fel a következő lemma.

7. LEMMA. *Ha $n > 1$, akkor az n csúcsú gráfok száma megadható az $n - 1$ csúcsú helyreállítható foksorozatok és az n hosszú helyreállítható nullmentes sorozatok összegeként, azaz*

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

Bizonyítás. Lásd [13, 21]. □

Ennek következményeképp a G_n érték kiszámítható az azt megelőző G értékek felhasználásával.

3. KÖVETKEZMÉNY. *Ha $n \geq 1$, akkor az n csúcsú egyszerű gráfok foksorozatainak száma megadható a $2 \leq i \leq n$ csúcsú egyszerű gráfok nullmentes foksorozatainak felhasználásával a következő módon:*

$$G_n = 1 + \sum_{i=2}^n Z_n \tag{30}$$

Bizonyítás. Lásd [13]. □

A program működésének leírása során dőlt kisbetűk jelölik a felhasznált változókat és dőlt nagybetűk a konstansokat, valamint a fontosabb hálózati primitíveket.

A programot négy részben ismertetjük, ezek pedig a következők: a 4.1. részben leírjuk a szerver program működését, amit folyamatábrák segítségével ismertetünk,

a 4.2. részben pszeudokóddal ismertetjük a szerver feladatokat kiosztó részét, azaz eltekintünk a hálózati kommunikációval kapcsolatos részek részletezésétől. A 4.3. részben a kliens működését írjuk le folyamatábrákkal, majd ezt a program grafikus sorozatokat leszámlláló részének ismertetése követi a 4.4. részben, pszeudokódok felhasználásával.

4.1. Kiszolgáló program

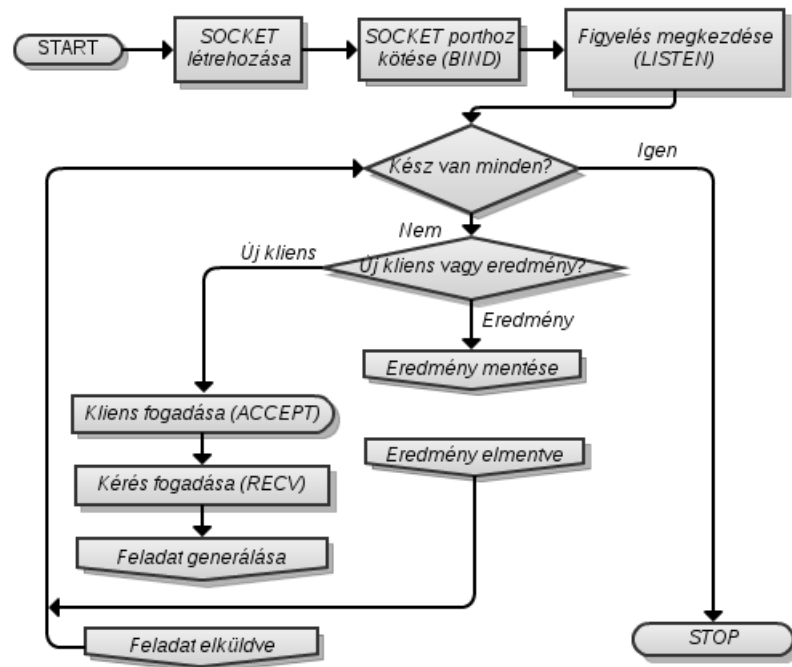
Programunkban egy központi kiszolgáló program osztja ki a részfeladatokat a számításokhoz csatlakozó kliensek között. Ennek készítésekor az elsődleges szempontunk a lehető legnagyobb megbízhatóság, rugalmasság és sebesség elérése volt.

Megebízhatóságon azt értjük, hogy a rendszer működését külső természeti vagy akár humán tényezők ne befolyásolják. A kiszolgáló program ezért bármikor leállítható és újraindítható anélkül, hogy ez eredmények elvesztésével járna. A szerver leállása a klienseket nem befolyásolja. Ez egyben azt is jelenti, hogy amennyiben szükséges, a szerver programot egy komolyabb számítási projekt során bármikor kicserélhetjük vagy megváltoztathatjuk.

A szerver rugalmassága abban rejlik, hogy működése semmilyen módon nem függ össze azzal, hogy a kliensekben mi történik, csupán annyi a teendője, hogy részfeladatokat oszt ki és eredményeket gyűjt be. Az sem jelent problémát, ha közben minden klienst megállítunk, mert ezzel a szerver egyáltalán nem foglalkozik. Ez elsőre ugyan érdektelennek tűnhet, de nem az, ha figyelembe vesszük, hogy a kliens bárki számára szabadon elérhető az interneten, így számítani kell arra, hogy valaki úgy dönt, hogy részt vesz egy komoly számításban és elindítja a klienst, majd meggondolja magát és mégis inkább leállítja azt. Ehelyett gondolhatunk akár egy egyszerű áramszünetre is. Tovább lépve, mivel nincs állandó hálózati kapcsolat a kiszolgáló és a többi gép között, így nincs limitálva a résztvevő számítógépek száma sem. Ha valaki elindít a számítógépén egy klienst, az kap egy részfeladatot és számol.

Harmadik szempontunk a sebesség volt, ezért programunkat C nyelven valósítottuk meg. Választásunk azért erre a nyelvre esett a kezdeti MATLAB programok helyett, mert a MATLAB ugyan egy nagyon kellemes és jól átlátható futási felületet biztosít, de méréseink alapján C-vel megközelítőleg százszoros gyorsulás érhető el. Programunk tehát C-ben készült, Berkeley Socketek felhasználásával. Működése pedig a következő:

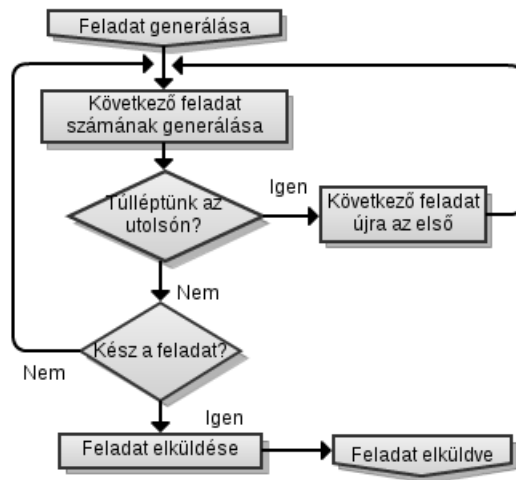
- A szerver indításakor létrehoz egy csatlakozót (*SOCKET*), amit aztán egy csatlakozási ponthoz (*PORT*) köt (*BIND*). A csatlakozási pont a programban előre definiált érték. Amint létrejött a csatlakozási pont és a szerver alkalmas a kapcsolatok fogadására, ellenőrzi, hogy van-e kész eredmény a naplófájlban. Amennyiben a napló nem üres, úgy az abban szereplő részfeladatokat késznek jelöli. Ezután elkezdődhet a kliensek bejövő kapcsolatainak kezelése (*LISTEN*).
- Mindaddig, amíg a befejezett részfeladatok száma (*finished_jobs*) el nem éri az



1. ábra. A szerver működésének folyamatábrája.

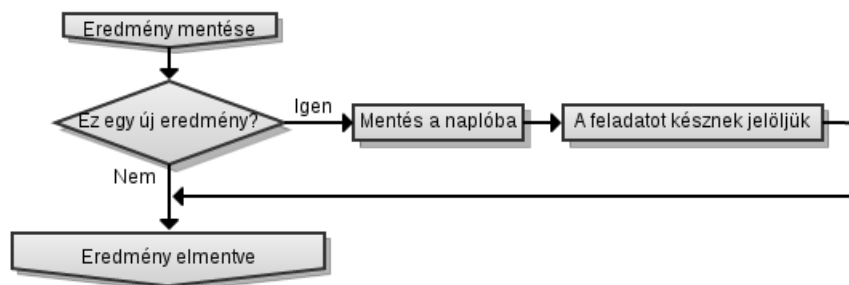
összes részfeladat számát (NUM_JOB), addig újra és újra klienseket fogadunk, majd a fogadott kliens üzenetéből eldöntjük, hogy mi célból jött. A kliens két okból csatlakozhatott: új részfeladatot kér vagy elkészült egy részeredménnyel és azt szeretné visszaküldeni.

- Ha a kliens új részfeladatot kér, akkor a 2. ábrán látható módon kiosztjuk neki a soron következőt a következő algoritmus alapján:
 - + Megnöveljük az éppen aktuális részfeladat sorszámát tároló értéket ($actual_job$), majd ellenőrizzük, hogy nem léptünk-e túl a lehetséges legnagyobb sorszámon (NUM_JOB), valamint hogy a részfeladatok státuszát tároló vektorunk (job_status) alapján nincs-e már teljesítve a megadott részfeladat.
 - + Addig ismételtetjük ezt a műveletet, amíg meg nem találtuk a kiosztandó részfeladatot.
 - + Miután a részfeladat kiküldésre került, lebontjuk a kapcsolatot. Itt megjegyezzük, hogy ha elértük az utolsó részfeladatot, akkor módszeresen elkezdjük újra kiosztani a már legrégebben kiadott részfeladatokat arra az esetre számítva, hogy a kliens, amely azt a részfeladatot végezte volna, valamilyen okból (számítógép leállítása, áramszünet stb.) nem tudta azt befejezni. Ez nyilván azt eredményezi, hogy amikor már nagyon kevés munka van vissza, akkor sok kliens fogja ugyanazt a részfeladatot



2. ábra. Új részfeladat kiosztása a kliensnek.

megkapni, így a részfeladatok befejeződésével egyre robusztusabbá válik a rendszer.



3. ábra. Részfeladat eredményének mentése.

- Ha a kliens eredményt küld, akkor azt a 3. ábrán látható algoritmus szerint a következő módon dolgozzuk fel:
 - + Fogadjuk az üzenetét és ellenőrizzük, hogy a beérkező eredmény nem szerepel-e már a teljesített részfeladatok listájában (*job_status*); amennyiben nem szerepel, úgy elmentjük és befejezettnek tekintjük.
 - + A mentés során a kliens által küldött üzenet eredeti formában kerül hozzáfűzésre a napló fájlhoz (*LOG_FILE*). A napló fájlban csv formátumban kerülnek rögzítésre az adatok. A fájl pontos formátumát a 4.3. részben ismertetjük.

4.2. Feladatokat kiosztó algoritmus (DJ)

Az előző részben ismertetett kiszolgáló program feladatokat kiosztó részét írjuk le ebben a részben, a feladatok kiosztására koncentrálva. Ebből kifolyólag a hálózati kommunikációval kapcsolatos részeket a most következő pszeudokód nem tartalmazza. A program teljes forráskódja elérhető a <http://people.inf.elte.hu/lulsaai/Holzhacker/> oldalon.

Bemenet. n : a gráf csúcsainak száma;

N : a részfeladatok száma;

M : a részfeladatok paramétereit tároló mátrix.

Kimenet. Z_n : Az n csúcsú nullmentes grafikus sorozatok száma.

Munkaváltozók. S : a részfeladatok státuszát tároló vektor;

fj : az elkészült részfeladatok száma;

aj : a legutóbb kiosztott részfeladat száma;

ji : bejövő részfeladat eredményének indexe;

c : kliens azonosító (a hálózati kommunikációnál kerül alkalmazásra);

msg : a klientsztől bejövő üzeneteket tárolja (szintén a hálózati kommunikációnál fontos).

DISTRIBUTING-JOBS(n, N, M)

```

01  $S_0 = true$                                 ▷ 01–05. sor: részfeladatok státuszát tároló vektor beállítása
02  $S_{N+1} = true$ 
03 for  $j = 1$  to  $N + 1$ 
04    $S_j = false$ 
05 end
06 while  $fj < N$                                 ▷ 06. sor: amíg el nem készül minden részfeladat
07   accept( $c$ )                                ▷ 07. sor: csatlakozó kliens fogadása
08   recv( $c, msg$ )                            ▷ 08. sor: üzenet fogadása a klientsztől
09   if  $msg == 0$                                 ▷ 09. sor: a kliens új feladatot kér
10      $aj = aj + 1$                             ▷ 10. sor: az aktuális feladat indexét növeljük
11     for  $i = M_{aj-1,0}$  to  $n$                 ▷ 11–13. sor: frissítjük a kezdősorozatot
12        $F_i = n + M_{aj-1,1}$ 
13     end
14     while  $S_{aj} == true$  or  $aj > N$         ▷ 14–22. sor: új feladat keresése
15        $aj = aj + 1$ 
16       if  $aj > N$                             ▷ 16. sor: túlléptünk a maximális indexen
17          $aj = 1$                                 ▷ 17. sor: visszalépünk az elsőhöz
18       end
19       for  $i = M_{aj-1,0}$  to  $n$                 ▷ 19–21. sor: kezdősorozat frissítése
20          $F_i = n + M_{aj-1,1}$ 
21       end
22   end

```

```

23      if  $aj < N$            ▷ 23–29. sor: feladat kilépési feltételeinek beállítása
24           $a = M_{aj,0}$ 
25           $b = n + M_{aj,1}$ 
26      else
27           $a = 1$ 
28           $b = 1$ 
29      end
30      send( $c, F, a, b$ )      ▷ 30. sor: részfeladat elküldése a kliensnek
31  else
32      recv( $c, ji, F_{init}, F_{last}, z_{n,m}, time$ )  ▷ 32. sor: eredmények fogadása
33      if  $S_{ji} == false$       ▷ 33. sor: ha a feladat eddig nem volt kész
34           $S_j = true$            ▷ 34. sor: készre állítjuk
35           $fj = fj + 1$          ▷ 35. sor: kész feladatok számát növeljük
36           $Z_n = Z_n + z_{n,m}$    ▷ 36. sor:  $Z_n$  részeredmény számítása
37      end
38  end
39      close( $c$ )              ▷ 39. sor: hálózati kapcsolat bontása
40 end
41 return  $Z_n$                 ▷ 41. sor: eredmények visszaadása

```

4.3. Kliens program

A kliens program elkészítésekor a fő cél a lehető legnagyobb egyszerűség elérése volt. Olyan programot akartunk készíteni, amely nem igényel semmiféle felhasználói beavatkozást, beállítást vagy akármi mást az elindításán kívül. Ez azért fontos, mert a számításokat a lehető legtöbb számítógép között szeretnénk szétosztani, tekintettel az óriási futási időkre.

Fontos volt az is, hogy ha egy komolyabb számítás befejeződik, akkor se kelljen minden egyes gépen újra és újra elindítani a programot, hanem az a lehető legkisebb erőforrás igénytel maradjon a háttérben és várjon mindaddig, amíg részt nem vehet egy újabb számításban. Ez sokat segített, amikor az új G érték számítását végeztük több mint 200 számítógép felhasználásával. Ezek között a gépek között voltak olyanok, ahol a processzorban négy mag is volt, így egyszerre négy klienst kellett volna újraindítani, amikor az egyik G érték kiszámítását követően áttértünk egy nagyobbra.

Jogosan felmerül a kérdés, hogy miért nem elegendő egyetlen kliens futtatása több szálon gépenként. A számításokat nem csak laborgépekkel végeztük, hanem programunkat publikusan elérhetővé tettük az interneten, így bárki csatlakozhatott, ezért nem szerettük volna, hogy valaki, aki egy klienst futtatva a gépen segíthetné munkánkat, azért ne használja azt, mert az teljesen leterheli a számítógépét.

A kliens program működése a következő:

- Miután létrehoztuk a csatlakozáshoz szükséges csatlakozót (*SOCKET*), megpróbálunk csatlakozni a kiszolgáló programhoz. Ha ez nem lehetséges, mert a szerver nem elérhető, akkor *wait.time* ideig várakozunk és egyúttal

feladat sorszáma	első sorozat		
	utolsó sorozat	futási idő	grafikus sorozatok száma

2. táblázat. A kliens által küldött eredmények szerkezete.

megkészserezzük a várakozási időt. (A *wait_time* a programban előre definiált érték. Alapértelmezése *DEFAULT_WAIT_TIME*.) Ezt addig ismételtetjük, amíg fel nem épül a kapcsolat vagy el nem érjük a *MAX_WAIT_TIME* értéket. Ezt követően már nem növeljük tovább a várakozás időtartamát. A módszer alapötletét Jacobson 1988-as [16], lassú kezdést biztosító protokollja adta. Megjegyezzük, hogy a neve ellenére a módszer egyáltalán nem lassú, hiszen exponenciális növekedést eredményez.

- A kapcsolat létrejöttét követően üzenetet küldünk a szerver programnak, amellyel egy részfeladatot kérünk tőle, majd miután megkaptuk azt, lebontjuk a hálózati kapcsolatot és a 4.4. részben ismertetett algoritmus segítségével előállítjuk és ellenőrizzük a kiosztott részfeladatba tartozó sorozatokat.
- A kliens az ellenőrzések során csak olyan sorozatokat állít elő és ellenőriz, amelyek párosak és nullmentesek. Ez alól egyetlen kivétel lehetséges: páratlan n értékek esetén az utolsó ellenőrzött sor nem az $1, 1, \dots, 1$, hanem a $-1, -1, \dots, -1$, mivel a sorozatokat generáló algoritmus kettesével lépteti a sorozatok tagjait anélkül, hogy 0 előfordulna, így az utolsó előtti ellenőrzött sorozat a $2, 1, 1, \dots, 1$, amely a következő lépésben – mivel nulla nem lehet – $-1, -1, \dots, -1$ -re vált. Ez azonban nem jelent problémát, mert az ellenőrző algoritmus elutasítja.
- A számítások végeztével a kapott eredményeket elküldjük a szervernek, amely aztán azokat a 2. táblázatban látható formátumban menti. A táblázat üres cellái későbbi felhasználásra vannak fenntartva.
- A kiszámított eredmények elküldése is hasonló módon történik, mint ahogy a részfeladatot kértük a szervertől. Azonban most nem próbálkozunk a végtelenségig. Összesen *MAX_RECONNECT_TRY* alkalmat követően abbahagyjuk a próbálkozást. Ez azért fontos, mert ha a szerver megkapott minden részeredményt, leáll, de a kliensek tovább dolgoznak. Ezzel az elsődleges célunk, hogy ne kelljen a klienseinket minden egyes projekt esetén újra és újra elindítani mindenütt, hanem ha a szerver huzamosabb ideig nem érhető el, akkor dobják el részeredményeiket és folytassák a normális működést, azaz kérjenek új részfeladatot, pontosabban várakozzanak új részfeladatra. Ezzel ismét csak a rendszer használatának egyszerűsítését és nagyobb hibatűrést akartunk elérni.

4.4. Felsoroló Erdős-Gallai algoritmus (EGE)

A 2.5. részben ismertetett lineáris idejű algoritmust felhasználva megadható egy olyan algoritmus, amely lexikografikus sorrendben előállítja az összes nullmentes, páros sorozatot és ellenőrzi azokat. Ez az algoritmus képezi a 4.3. részben ismertetett program alapját, még hozzá úgy, hogy az előző részben ismertetett program részeként párhuzamosan több számítógépen, több példányban fut és számol. Az algoritmus megtalálható a [12] cikkünkben, angol nyelven.

Az algoritmust a könnyebb érthetőség érdekében két részre bontottuk: az első rész (ERDŐS-GALLAI-CHECK) a sorozat ellenőrzését elvégző algoritmust, a második pedig (ERDŐS-GALLAI-ENUMERATING) az ellenőrizendő sorozatok előállítását és a hozzájuk kapcsolódó további értékek (F, H, c, D) frissítését végző programot.

Bemenet. $F = (f_1, \dots, f_n)$: ellenőrizendő sorozat;
 H : első i elem összegét tartalmazó értékek;
 c : ellenőrzőpontok száma;
 D : ellenőrzőpontok.

Munkaváltozó. J : dinamikusan változó méretű vektor, ami a súlypontokat tárolja.

Kimenet. L : logikai érték, ami *true*, ha a sorozat helyreállítható, és *false*, ha a sorozatot nem lehet helyreállítani.

ERDŐS-GALLAI-CHECK(F, H, c, D)

01 $i = 1$	▷ 01. sor: i kezdőértékének beállítása
02 while $i < c$ and $H_i > i(i - 1)$	▷ 02–15. sor: sorozat ellenőrzése
03 $p = D_i$	▷ 03. sor: kezdeti súlypont
04 while $J_p < n$ and $f_{J_p+1} > p$	▷ 04–09. sor: súlypont frissítése
05 $J_p = J_p + 1$	
06 end	
07 while $J_p > p$ and $f_{J_p} \leq p$	
08 $J_p = J_p - 1$	
09 end	
10 if $H_p > H_n - H_{J_p} + p(J_p - 1)$	▷ 10. sor: ellenőrzés
11 $L = false$	▷ 12. sor: rossz sorozat
12 return L	
13 end	
14 $i = i + 1$	
15 end	
16 $L = true$	▷ 17. sor: helyreállítható sorozat
17 return L	

Bemenet. n : a csúcsok száma;
 F : az első részfeladat esetében a kezdősorozat, egyébként az aktuális részfeladatot

megelőző feladat utolsó ellenőrzött sorozata;
 a : az utolsó sorozat azonosítására használt index;
 b : az utolsó sorozat a -adik elemének értéke.

Kimenet. $z_{n,m}$: a Z_n m -edik részletösszege (Z_n jelöli azoknak a helyreállítható sorozatoknak a számát, amelyek nem tartalmaznak nullát, azaz *nullmentesek*).

ERDŐS-GALLAI-ENUMERATING(n, F, a, b)

```

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  $f_n \neq n - 1$                             ▷ 05. sor: ha nem a teljes gráf foksorozata
06     if  $H_n$  páratlan                            ▷ 06–12. sor: aktualizálás
07          $f_n = f_n - 3$ 
08          $H_n = H_n - 3$ 
09     else
10          $f_n = f_n - 2$ 
11          $H_n = H_n - 2$ 
12     end
13 end
14 for  $i = 1$  to  $n$                             ▷ 14–16. sor: súlypontok előkészítése
15      $J_i = n - 1$ 
16 end
17  $c = 0$                                 ▷ 17. sor: ellenőrzőpontok számának beállítása
18 for  $i = 1$  to  $n - 2$                             ▷ 18–23. sor: ellenőrzőpontok meghatározása
19     if  $f_i \neq f_{i+1}$  and  $f_i \neq f_n$ 
20          $c = c + 1$                                 ▷ 20. sor: ellenőrzőpontok számának növelése
21          $D_c = i$ 
22     end
23 end
24  $L = \text{ERDŐS-GALLAI-CHECK}(F, H, c, D)$     ▷ 24. sor: első sorozat ellenőrzése
25  $z_{n,m} = z_{n,m} + L$ 
26 while  $f_a > b$                             ▷ 26. sor: amíg el nem érjük az utolsó ellenőrizendő sort
27      $k = n$                                 ▷ 27. sor: munkaváltozó inicializálása
28     if  $f_k == 1$                             ▷ 28. sor: ha a sorozat 1-re végződik
29          $j = n - 1$                             ▷ 29–32. sor: nem 1 tagjainak meghatározása
30         while  $f_j \leq 1$ 
31              $j = j - 1$ 
32         end
33     if  $f_j == 2$     ▷ 33. sor: ha az 1-et nem tartalmazó rész 2-re végződik
34          $f_{j-1} = f_{j-1} - 1$                 ▷ 34. sor: a sorozat frissítése
35          $H_{j-1} = H_{j-1} - 1$                 ▷ 35. sor:  $H$  értékek frissítése
36     if  $j > 2$                                 ▷ 36–49. sor: ellenőrzőpontok frissítése

```

```

37         if ( $c \leq 2$  or ( $c > 2$  and  $D_{c-2} \neq j - 2$ )) and
(c > 1 and  $D_{c-1} \neq j - 2$ )
38             if  $c > 1$  and  $D_{c-1} > j - 2$ 
39                  $D_{c+1} = D_c$ 
40                  $D_c = D_{c-1}$ 
41                  $D_{c-1} = j - 2$ 
42                  $c = c + 1$ 
43             else
44                  $D_{c+1} = D_c$ 
45                  $D_c = j - 2$ 
46                  $c = c + 1$ 
47             end
48         end
49     end
50     for  $k = j$  to  $n$ 
51          $f_k = f_{j-1}$  ▷ 51. sor: a sorozat végének frissítése
52          $H_k = H_{k-1} + f_k$  ▷ 52. sor:  $H$  értékek frissítése
53     end
54     while  $c > 1$  and  $D_c > j - 1$  ▷ 54–56. sor: ellenőrzőpontok
55          $c = c - 1$ 
56     end
57     if  $H_n$  páratlan ▷ 57. sor: rossz paritás esetén
58          $f_n = f_{n-1} - 1$  ▷ 58. sor: sorozat frissítése
59          $H_n = H_{n-1} + f_n$  ▷ 59. sor:  $H$  számítása
60          $c = c + 1$  ▷ 60–61. sor: ellenőrzőpontok bővítése
61          $D_c = n - 1$ 
62     end
63     else
64          $f_j = f_j - 1$  ▷ 64. sor: sorozat frissítése
65          $H_j = H_j - 1$  ▷ 63. sor:  $H$  frissítése
66         if  $j > 1$  ▷ 66–74. sor: ellenőrzőpontok frissítése
67             if ( $c == 1$  and  $D_c \neq j - 1$ ) or ( $c > 1$  and  $D_{c-1} \neq j - 1$ )
68                 if  $c > 0$  and  $D_c > j - 1$ 
69                      $D_{c+1} = D_c$ 
70                      $D_c = j - 1$ 
71                      $c = c + 1$ 
72                 end
73             end
74         end
75         for  $k = j + 1$  to  $n$ 
76              $f_k = f_j$  ▷ 76. sor: sorozat frissítése
77              $H_k = H_{k-1} + f_k$  ▷ 77. sor:  $H$  számítása
78         end
79         while  $c > 1$  and  $D_c > j - 1$  ▷ 79–81. sor: ellenőrzőpontok

```

```

80              $c = c - 1$ 
81         end
82         if  $H_n$  páratlan ▷ 82. sor: paritás ellenőrzése
83              $f_n = f_n - 1$  ▷ 83. sor: sorozat frissítése
84              $H_n = H_n - 1$  ▷ 84. sor:  $H$  számítása
85              $c = c + 1$  ▷ 85. sor: ellenőrzőpontok számának növelése
86              $D_c = n - 1$  ▷ 86. sor: új ellenőrzőpont hozzáadása
87         end
88     end
89 else
90     if  $f_k == 2$ 
91          $f_{k-1} = f_{k-1} - 1$  ▷ 91. sor: sorozat frissítése
92          $H_{k-1} = H_{k-1} - 1$  ▷ 92. sor:  $H$  frissítése
93         if ( $c == 1$  and  $D_c \neq n - 2$ ) ▷ 93–101. sor: ellenőrzőpontok
94             or ( $c > 1$  and  $D_{c-1} \neq n - 2$  and  $D_c \neq n - 2$ )
95             if  $c > 0$  and  $D_c > n - 2$ 
96                  $D_{c+1} = D_c$ 
97                  $D_c = n - 2$ 
98             else
99                  $c = c + 1$ 
100                  $D_c = n - 2$ 
101             end
102         end
103         if  $f_{k-1}$  páratlan ▷ 102. sor: paritás ellenőrzése
104              $f_k = f_{k-1}$  ▷ 103. sor: sorozat frissítése
105             if  $c > 0$  and  $D_c == n - 1$ 
106                  $c = c - 1$  ▷ 105. sor: ellenőrzőpontok
107             end
108             else
109                  $f_k = f_{k-1} - 1$  ▷ 108. sor: sorozat frissítése
110             end
111              $H_k = H_{k-1} + f_k$  ▷ 110. sor:  $H$  számítása
112         else
113              $f_k = f_k - 2$  ▷ 112. sor: sorozat frissítése
114              $H_k = H_k - 2$  ▷ 113. sor:  $H$  számítása
115             if  $c < 1$  or  $D_c \neq n - 1$  ▷ 114–117. sor: ellenőrzőpontok
116                  $c = c + 1$ 
117                  $D_c = n - 1$ 
118             end
119         end
120     end
121 end
122      $z_{n,m} = z_{n,m} + \text{ERDŐS-GALLAI-CHECK}(F, H, c, D)$  ▷ 120. sor: ellenőrzés
123 end

```

Ez az algoritmus lehetővé teszi, hogy egyszerre párhuzamosan több géppel

végezzük a Z értékek kiszámítását, úgy, hogy a kezdősorozattól az utolsó sorozatig minden sorozatot tesztelünk. Az utolsó sorozatot – a kezdő sorozattal ellentétben – nem adjuk meg bemenetként, csupán az azonosításához használt két értéket: a , b . Ezekkel az értékekkel a következő módon azonosítható a sorozat: mivel a sorozatokat lexikografikus sorrendben állítjuk elő, a sorozat egyetlen elemének ellenőrzésével (f_a) eldönthető, hogy elértük-e már az utolsó ellenőrizendő sorozatot, azaz teljesül-e, hogy $f_a < b$. Ha igen, akkor a kiosztott részfeladatot elvégzettnek tekintjük.

5. A feladat részekre bontása

A programok működését korábban már ismertettük. Most leírjuk azt is, hogy a problémát milyen technikák segítségével osztottuk fel a számítógépek között. Elsőként leírjuk a legegyszerűbb naiv megközelítést, amely kisebb n értékek esetén még megfelelő, majd áttérünk arra, hogy nagyobb n -eknél milyen módszerekkel próbáltuk kiegyensúlyozni az egyes számítógépek terhelését.

5.1. Naiv megközelítés

Legelső megközelítésünk szerint a problémát kisebb n értékek esetén tapasztalt futási idők alapján igyekeztünk azonos méretű darabokra osztani. Ez G_{24} esetén még elfogadható eltéréseket eredményezett, de sajnos nagyobb n -ekre már olyan mértékű lett volna az eltérés, ami rontotta volna a párhuzamosítás hatékonyságát.

Ezt a megközelítést felhasználva a problémát összesen 23 részproblémára (*szelet*, *job*) bontottuk. A futási időkre vonatkozó adataink közlésétől eltekintünk, tekintve, hogy a részfeladatok több, egymástól eltérő hardverrel felszerelt számítógépen futottak.

5.2. Felosztás korábbi n értékekre alapozva

Az előző megközelítés legnagyobb hiányossága a rendszertelenség volt, hiszen se az összesen megvizsgált sorozatok számát, sem az elfogadott sorozatokat nem vette számításba. Az elfogadott sorozatok számát fontos tényezőként kell kezelni, hiszen ezekben az esetekben a teszt minden egyes ellenőrzőpontban lefut, így ezek a tesztek veszik igénybe a legtöbb időt.

A sorozatok szerinti felosztást a következő módon generáltuk. A korábban sorozatok generálására felhasznált programunkat úgy alakítottuk át, hogy a tesztet eltávolítottuk, csupán azt figyeltük, hogy hány sorozaton vagyunk túl. Mivel az összes és a páros sorozatok számát ismertük, így könnyedén meghatározható volt, hogy mikor vagyunk két lehetséges részfeladat határán. A kapott sorozatot teljes egészében tárolni azonban nem lett volna célszerű, hiszen az rendkívül nagy mértékben lassítaná a számolást, ha minden egyes sorozat ellenőrzését követően ellenőriznünk kellene az éppen aktuális sorozat minden elemét, hogy nem értük-e el az utolsó ellenőrizendő sort. A kapott sorozatokról tehát csak annyi információt tároltunk, hogy mi az első elem indexe, ami eltér az azt megelőző sorozattól, és

hogy mi ez az érték. Ezeket az információkat felhasználva már elegendő egyetlen ellenőrzést elvégezni, hogy eldönthessük, végeztünk-e a kapott részfeladattal.

Ez a „csonkolás” azonban újra csak azt eredményezi, hogy a részfeladatok eltérő méretűek lesznek. Ez a probléma aztán egyre nagyobb eltéréseket okoz, ha az n érték növekszik.

Ezzel a technikával egy 435 részfeladatot tartalmazó felosztást készítettünk és használtunk fel a Z_{25} és Z_{26} értékek számításához. Ezekben az esetekben már nem a G értékeket számítottuk. Ennek oka, hogy a 7. lemma alapján G értékét könnyedén meghatározhatjuk Z ismeretében és így a szükséges számolási idő a szabályos bemenetekhez tartozó idő $\frac{1}{4}$ -ére csökken [13].

5.3. Felosztás a műveletszámok alapján

Az előző megoldásunk már viszonylag jó eloszlást biztosított a futási idők tekintetében, azonban előfordul benne 2–3 óriás részfeladat is, amelyek megoldása akár tízszer annyi ideig is tartott, mint a többieké. Ezt a jelenséget az alábbi módon próbáltuk meg kiküszöbölni.

A generált részsorozatok már nem futási idejük vagy az ellenőrzött sorozatok száma alapján osztottuk fel, hanem az ellenőrzések során elvégzett műveletek száma adta a sorozatok közti határokat. Ezt úgy hajtottuk végre, hogy egy alacsonyabb n értékre kiszámítottuk a teljes műveletigényt, majd ennek ismeretében generáltuk az új részfeladatokat. Nyilván ennek a felosztásnak is velejárója volt, hogy sorozatainkat „csonkoltuk” a nagyobb sebesség elérésének érdekében.

A Z_{28} értékét ezt a felosztást felhasználva számítottuk ki.

5.4. Felosztás a páros sorozatok száma alapján

A korábbi megvalósítások ugyan közelítőleg megfelelő felosztást eredményeztek, azonban mindegyikben fordultak elő – még ha csak kis számban is – mamutok (az átlagos futási idő többszörösét igénylő részfeladatok). Ennek elkerüléséhez hasznos a (15) egyenlet.

A (15) egyenletből kiindulva kiszámítható, hogy hány olyan sorozat létezik, amelyeknek m -edik tagja p . Ezt felhasználva megadható egy mátrix, amely megadja, hogy közelítőleg hány olyan sorozat van, amely az f_1 értékkel kezdődik és f_2 értékkel folytatódik stb. Ezt a táblázatot felhasználva a következő módon adható meg a felosztás:

- Válasszuk meg a maximális részfeladat méretet (esetünkben ez akkora szeleteket jelentett, amelyek egy éjszaka alatt kiszámíthatóak).
- Induljunk el a mátrix alsó sorának első elemétől és kezdjük a mátrix további sorait kiolvasni és összegezni mindaddig, amíg az aktuális kapacitás vagy a kiolvasandó értékek el nem fogytak.
- Ha egy érték túl nagy az adott maximális mérethez viszonyítva, akkor lépünk egy sorral feljebb és kezdjük el azt a sort kiolvasni addig az oszlopig, ahonnan

1	1	1	1
1	2	3	4
1	3	6	10
1	4	10	20
1	5	15	35

3. táblázat. A felosztáshoz használt mátrix $n = 5$ esetén.

2, 2, 2, 2, 2
3, 2, 2, 2, 2
3, 3, 3, 3, 3
4, 2, 2, 2, 2
4, 3, 3, 3, 3
4, 4, 3, 3, 3
4, 4, 4, 4, 4

4. táblázat. $n = 5$ -ös felosztás határsorozatai.

felléptünk.

- Folytassuk ezt az algoritmust, amíg az utolsó sor végére nem értünk.

A korábban ismertetett algoritmus szerint $n = 5$ -re a 3. táblázatból kiolvasható a 4. táblázatban látható felosztás.

A táblázatban látható sorozatok a következő módon értendők: az első szeptet az 1, 1, 1, 1, 1 sorozattól a 2, 2, 2, 1, 1 sorozatig tart, a második pedig a 2, 2, 2, 2, 2 sorozattól a 3, 2, 2, 1, 1 sorozatig és így tovább.

Ez a módszer ugyan nem garantálja, hogy a részfeladatok hossza pontosan ugyanakkora legyen, azonban megszünteti a korábbiakban tapasztalt „mamut” hatást.

Ezt a módszert használtuk fel Z_{29} kiszámításakor. A teljes számítást összesen több mint 15.000 részfeladatra felosztva végeztük.

A teljes felosztás generálásának első lépése a 3. táblázathoz hasonló mátrix előállítása a megfelelő n értékhez. A mátrix feltöltését végzi el az alábbi program.

Bemenet. n : a csúcsok száma ($n \geq 4$);

MaxSize: a maximális megengedett „szeletméret” (nullmentes páros sorozatok maximális száma a szeletben).

Kimenet. A képernyőre írjuk a szeleteket határoló sorozatokat.

GENMATRIX(n , $MaxSize$)

```

01 for  $j = n - 1$  downto 1           ▷ 01–03. sor: a mátrix első sorának kitöltése
02      $M_{1,j} = 1$ 
03 end

```

```

04 for  $i = n$  downto 2                                ▷ 04–08. sor: a mátrix feltöltése
05     for  $j = 1$  to  $n - 1$ 
06          $M_{i,j} = \binom{i+j-2}{i-1}$ 
07     end
08 end
09 GENSEQUENCES( $M, n, n, 1, n - 1, MaxSize, 0$ )          ▷ 09. sor: szelet generálás

```

Miután megvan a mátrixunk, már csak ki kell olvasnunk belőle a szeletek határsorozatait (a 4. táblázathoz hasonló alakban) az alábbi algoritmus segítségével.

Bemenet. M : a felosztáshoz tartozó mátrix, amit a GENMATRIX algoritmussal kaptunk;

n : a csúcsok száma ($n \geq 1$);

i, j, J : rekurzív segédparaméterek;

$MaxSize$: a maximális megengedett szeletméret (nullmentes páros sorozatok maximális száma a szeletben).

Kimenet. A szeleteket határoló sorozatok (rekurzív).

```

GENSEQUENCES( $M, n, i, j, j_m, MaxSize, J$ )
01  $C = 0$                                                 ▷ 01. sor: a szelet méretének inicializálása
02 while  $j < j_m + 1$ 
03     if  $C + M_{i,j} \leq MaxSize$                         ▷ 03. sor: ha bővíthető a szelet
04          $C = C + M_{i,j}$                                 ▷ 04. sor: bővítés
05     if  $j \leq j_m$                                     ▷ 05–07. sor: tovább lépünk a mátrixban
06          $j = j + 1$ 
07     end
08     else                                              ▷ 08. sor: nem bővíthető a szelet
09         if  $C \neq 0$                                     ▷ 09. sor: a szelet nem üres
10             for  $k = 2$  to  $size(J, 2)$                 ▷ 10–16. sor: kiírás
11                  $print(J_k)$ 
12             end
13             for  $k = 1$  to  $n - size(J, 2) + 1$ 
14                  $print(j - 1)$ 
15             end
16              $print\ newline$                             ▷ 16. sor: sortörés
17              $C = 0$ 
18         end
19         if  $M_{i,j} > MaxSize$  and  $j \leq j_m$           ▷ 19. sor: felbonthatóság ellenőrzése
20             GENSEQUENCES( $M, n, i - 1, 1, j, MaxSize, [J, j]$ )
21              $j = j + 1$ 
22         end
23     end
24 end
25 if  $C \neq 0$                                           ▷ 25. sor: az utolsó szelet nem üres

```

```

26  for  $k = 2$  to  $size(J, 2)$                                 ▷ 26–32. sor: utolsó szelet kiírása
27      print( $J_k$ )
28  end
29  for  $k = 1$  to  $n - size(J, 2) + 1$ 
30      print( $J(size(J, 2))$ )
31  end
32  print newline
33 end

```

6. Eredmények

A korábban ismertetett Holzhacker program segítségével bővítettük a *The On-Line Encyclopedia of Integer Sequences* [24] adatbázist, amely G_{23} -ig tartalmazta a foksorozatok számát. Programunk segítségével megadtuk a hiányzó értékeket egészen G_{29} -ig. Ezek az értékek megtalálhatóak a 5. táblázatban (az új értékek félkövérek).

Számításaink során igénybe vettünk laborgépeket (Komputeralgebra labor, Nyelvi labor, Adatbázis labor, Lovarda) valamint magánszemélyek segítségét is. A számítógépek számítási teljesítményét összegezve azt mondhatjuk, hogy számítási kapacitásunk elméleti maximuma elérte a 6 TFLOPS sebességet. Részletesebb adatok találhatóak a 6. táblázatban. A táblázat helyenként hiányos, mivel a gyártók honlapjain sajnos nem minden processzor típus esetén érhetőek el a táblázatban szereplő teljesítmény adatok.

A számítások során létrejött naplófájlok a forráskódokkal együtt megtalálhatóak a honlapomon (<http://people.inf.elte.hu/lulsaai/Holzhacker>).

Az általunk kiszámított G_n értékek naplói közül a legrövidebbet tartalmazza a 7. táblázat rövidített formában, azaz nem tartalmazza a szeletek kezdő és befejező sorozatait, csak a sorszámot, futási időt és az elfogadott sorozatok számát. Itt megjegyezzük, hogy ebben a számításban még nem vettek részt a laborok gépei.

A különböző Z_n értékek részfeladatainak futási időt összegezve megkaphatjuk, hogy a teljes számítás egy géppel mennyi időt vette volna igénybe. Ezeket az adatokat tartalmazza a 8. táblázat.

7. Összefoglalás

A dolgozat témája gráfok foksorozatainak gyors tesztelése. A klasszikus módszerek ismertetése mellett megadtunk egy új, lineáris futási idejű módszert, bemutattunk egy foksorozatokat leszámoló párhuzamos programot és az azzal elért eredményeinket.

Az első részben röviden bemutattuk az alap problémát, annak alkalmazási területeit és a dolgozatban elérni kívánt célokat.

A második részben ismertettük az eredeti Erdős-Gallai módszert és a Tripathi és Vijay által megadott gyorsításokat. Ezután leírtuk a LINEÁRIS-ERDŐS-GALLAI-

n	R_n	E_n	G_n
1	1	1	1
2	3	2	2
3	10	6	4
4	35	19	11
5	126	66	31
6	462	236	102
7	1 716	868	342
8	6 435	3 235	1 213
9	24 310	12 190	4 361
10	92 378	46 252	16 016
11	352 716	176 484	59 348
12	1 352 078	676 270	222 117
13	5 200 300	2 600 612	836 315
14	20 058 300	10 030 008	3 166 852
15	77 558 760	38 781 096	12 042 620
16	300 540 195	150 273 315	45 967 479
17	1 166 803 110	583 407 990	176 005 709
18	4 537 567 650	2 268 795 980	675 759 564
19	17 672 631 900	8 836 340 260	2 600 672 458
20	68 923 264 410	34 461 678 394	10 029 832 754
21	269 128 937 220	134 564 560 988	38 753 710 486
22	1 052 049 481 860	526 024 917 288	149 990 133 774
23	4 116 715 363 800	2 058 358 034 616	581 393 603 996
24	16 123 801 841 550	8 061 901 596 814	2 256 710 139 346
25	63 205 303 218 876	31 602 652 961 516	8 770 547 818 956
26	247 959 266 474 052	123 979 635 837 176	34 125 389 919 850
27	973 469 712 824 056	486 734 861 612 328	132 919 443 189 544
28	3 824 345 300 380 220	1 912 172 660 219 260	518 232 001 761 434
29	15 033 633 249 770 520	7 516 816 644 943 559	2 022 337 118 015 338

5. táblázat. Szabályos, páros és helyreállítható sorozatok száma.

Labor / Tulajdonos	Gép	Processzor	Mag	GFLOPS
Adatbázis labor	60	i5-650	2	25.6
Komputeralgebra labor	20	E 5300	2	20.8
Lovarda	110	E 7500	2	23.4
Nyelvi labor	60	i5-650	2	25.6
Ballagi Dániel	1	E7300	2	21.2
Kása Zoltán	1	875P	?	?
Kása Zoltán	1	i7-2600	4	70.3
Mányoki Ádám	2	E5405	?	?
Mányoki Ádám	1	Q6600	4	38.4
Mányoki Ádám	1	X4 920	4	52.9
Méreg Balázs	1	T4500	2	?

6. táblázat. A számításaink során felhasznált számítógépek adatai.

Sorszám	Futási idő	$G_{24,m}$
1	886m38.590s	10 029 832 755
2	936m21.676s	28 723 877 732
3	505m47.829s	41 658 148 450
4	547m59.627s	69 578 274 838
5	511m25.114s	58 589 929 879
6	508m55.076s	102 412 209 899
7	542m10.017s	118 800 436 447
8	488m0.130s	151 600 893 997
9	643m54.655s	78 547 565 915
10	1518m44.534s	147 524 314 179
11	776m9.764s	70 426 738 550
12	585m91.666s	99 949 294 037
13	429m36.930s	49 782 771 161
14	429m26.178s	58 135 313 980
15	396m12.822s	118 914 450 088
16	375m33.124s	80 117 065 415
17	462m35.083s	115 347 245 437
18	471m41s	56 764 861 564
19	697m17.387s	207 784 332 910
20	565m3.692s	133 860 267 289
21	393m20.431s	151 426 892 693
22	549m59.291s	158 301 007 132
23	678m19,125s	148 434 414 999

7. táblázat. A G_{24} számításának rövidített naplója.

n	Futási idő (nap)	Részfeladatok száma
25	26	435
26	70	435
27	316	435
28	1130	2 001
29	6733	15 119

8. táblázat. A számítások során mért futási idők összege és a részfeladatok száma.

UGRÓ algoritmust, amit a későbbiekben a gráfsorozatokat leszámoló programban felhasználtunk.

A harmadik részben ismertettünk néhány eredményt az összes n -szabályos és n -páros sorozat számával és azok arányával kapcsolatban.

A dolgozat negyedik részében adtuk meg a Holzhacker program leírását, kitérve a program főbb sajátosságaira, valamint a megvalósítását meghatározó főbb irányelvekre és célokra. Ebben a részben ismertettük a szerver és a kliens program leszámolással kapcsolatos részeinek pszeudokódjait is, azaz a DISTRIBUTE-JOBS és az ERDŐS-GALLA-ENUMERATING algoritmusokat.

Az ötödik rész tartalmazta azokat a módszereket és algoritmusokat, amelyeket felhasználtunk az új Z és G értékek kiszámításakor.

A dolgozat hatodik részében ismertettük eredményeinket (az újonnan megadott Z_n és G_n értékeket, az új algoritmus és a régi algoritmus futási idejének összehasonlítását), valamint néhány technikai részletet a számítások elvégzésének körülményeiről (naplófájlok és elérhetőségük, felhasznált számítógépek típusa, rendelkezésre álló maximális számítási teljesítmény). Az ebben a részben ismertetett, általunk kiszámolt új G_n értékek 2011. november 15. óta megtalálhatóak az OEIS adatbázis A004251-es sorozatában [24].

A dolgozatban leírt eredményeket a [12, 13, 14] cikkekben publikáltuk.

Az eddig elért eredményeket szeretnénk a későbbiekben felhasználni focisorozatok [7, 22] és páros gráfok sorozatainak ellenőrzése során, mivel ezeknek a problémáknak része az egyszerű gráfok foksorozatainak ellenőrzése.

Hivatkozások

- [1] ASCHER, M.: *Mu torere: an analysis of a Maori game*. Math. Mag. **60**, (1987) 90–100. [⇒8](#)
- [2] BURNS, J. M.: *The Number of Degree Sequences of Graphs*. PhD Dissertation, MIT, 2007. [⇒10](#)
- [3] CHOUDUM, S. A.: *A simple proof of the Erdős-Gallai theorem on graph sequences*. Bull. Austral. Math. Soc. **33**, (1986) 67–70. [⇒5](#)

- [4] CHUNGPHAISAN, V.: *Conditions for sequences to be r -graphical*. Discrete Math. **7**, (1974) 31–39. [⇒3, 4](#)
- [5] ERDŐS, P., GALLAI, T.: *Gráfok előírt fokú pontokkal*. Mat. Lapok **11**, (1960) 264–274. [⇒3, 5](#)
- [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. [⇒3, 4](#)
- [7] FRANK, A.: *Connections in Combinatorial Optimization*. Calderon Oxford Univ. Press, Oxford, (2011) 640 p. [⇒29](#)
- [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. [⇒3, 4](#)
- [9] HAVEL, V.: *A remark on the existence of finite graphs (Czech)*. Časopis Pěst. Mat. **80**, (1955) 477–480. [⇒3, 4](#)
- [10] IVÁNYI, A.: *Reconstruction of complete interval tournaments*. Acta Univ. Sapientiae, Inform. **1**, (2009) 71–88. [⇒3, 4](#)
- [11] IVÁNYI, A.: *Reconstruction of complete interval tournaments. II*. Acta Univ. Sapientiae, Math. **2**, (2010) 47–71. [⇒3, 4](#)
- [12] IVÁNYI, A., LUCZ, L., PIRZADA, S.: *Parallel Erdős-Gallai algorithm*. CEJOR Cent. Eur. J. Oper. Res., (2011) benyújtva. [⇒18, 29](#)
- [13] IVÁNYI, A., LUCZ, L., MÓRI, F. T., SÓTÉR, P.: *On the Erdős-Gallai and Havel-Hakimi algorithms*. Acta Univ. Sapientiae, Inform. **3**(2), (2011) 230–268. Elérhető: <http://www.acta.sapientia.ro/acta-info/> [⇒3, 4, 8, 10, 11, 23, 29](#)
- [14] IVÁNYI, A., LUCZ, L., MÓRI, F. T.: *Lineáris Erdős-Gallai teszt*. Alk. Mat. Lapok., (2011) benyújtva. [⇒29](#)
- [15] IVÁNYI, A., PIRZADA, S.: *Comparison based ranking*. In (ed. A. Iványi): Algorithms of Informatics, **3**, AnTonCom, Budapest, 2011, 1209–1256. [⇒3](#)
- [16] JACOBSON, V.: *Congestion avoidance and control*. In: Proc. of SIGCOMM '88, Stanford, CA (1988), ACM, 314–329. [⇒17](#)
- [17] JÁRAI, A. (szerk.): *Bevezetés a matematikába*. ELTE Eötvös Kiadó, Budapest, (2005). [⇒7](#)
- [18] KLEITMAN, D. J., WANG, D. L.: *Algorithms for constructing graphs and digraphs with given valences and factors*. Disc. Mat., **6**, (1973) 79–88. [⇒3, 4](#)
- [19] LAMAR, M. D.: *Algorithms for realizing degree sequences of directed graphs*. arXiv (2009). Elérhető: <http://arxiv.org/abs/0906.0343> [⇒3, 4](#)

- [20] 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. [⇒3](#)
- [21] LUCZ, L., SÓTÉR, P.: *Fokszorozatok ellenőrző algoritmusok.* TDK dolgozat, ELTE IK, Budapest, 2011. [⇒3, 4, 5, 6, 7, 11](#)
- [22] LUCZ, L., IVÁNYI, A.: *Testing and enumeration of football sequences.* In (ed. Z. Csörnyei): Abstracts of MaCS 2012, Siófok, Február 9–12, 2012. [⇒29](#)
- [23] ÖZKAN, S.: *Generalization of the Erdős-Gallai inequality.* Ars Combin. **98** (2011) 295–302. [⇒3, 4](#)
- [24] 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/> [⇒3, 7, 26, 29](#)
- [25] SLOANE, N. J. A., PLOUFFE, S.: *The Encyclopedia of Integer Sequences.*, Academic Press, 1995. [⇒7, 8](#)
- [26] STANLEY, R. P.: *Enumerative Combinatorics*, Cambridge University Press, Cambridge, 1997. [⇒7](#)
- [27] TRIPATHI, A., VIJAY, S.: *A note on a theorem of Erdős & Gallai.* Disc. Math. **265**, (2003) 417–420. [⇒3, 5, 6](#)
- [28] 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. [⇒5](#)