Prhuzamos Erds-Gallai algoritmus

Lucz Lornd
Programtervez informatikus MSc
lorand.lucz@gmail.com

Ivnyi Antal
tmavezet
tony@compalg.inf.elte.hu

October 15, 2011

Abstract

Havel 1995-ben [5], Hakimi 1962-ben [4], Erds s Gallai 1960-ban [2], Tripathy, Venugopalan s West 2010-ben [17] javasoltak ngyzetes futsi idej mdszereket annak eldntsre, hogy egy nemnegatv szmokbl ll sorozat lehet-e egy egyszer grf foksorozata. 2011-ben Ivnyi, Lucz s Str [8] adta meg az Erds-Gallai algoritmus lineris idej vltozatt. Dolgozatunkban bemutatjuk ezeket az algoritmusokat, lerjuk a foksorozatok gyors ellenrzsnek mdszert, tovbb ismertetnk egy a grfsorozatok szmnak meghatrozsra ksztett szerver-kliens programot s az azzal elrt eredmnyeinket.

1 Bevezets

A bennnket krlvev vilgot sokflekppen modellezhetjk. Ennek egy lehetsges mdszere a vilg grfokkal trtn reprezentlsa. Valamilyen logika alapjn a krlttnk lev objektumokat logikailag sszekapcsolhatjuk, majd ezeket a kapcsolatokat knnyedn brzolhatjuk grfok segtsgvel. Egyszer pldaknt tekinthetnk pldul egy csaldft, ahol a grf egy cscsa jell egy szemlyt, a grf lei pedig a szl-gyermek viszonyokat. Ez alapjn mindenki a szleivel s gyermekeivel ll kzvetlen kapcsolatban. A grfok cscsai minden esetben legfeljebb egyszeresek lehetnek s a kapott grf hurokl mentes. sszefoglalva teht egy *egyszer grfot* kaptunk.

Mivel manapsg mr szinte mindent szmtgpekkel vgznk, ezrt fontos krds ezeknek a grfoknak a trolsa. Ennek egyik lehetsges mdja a cscsok foksz-mainak trolsa. Egy grfbl knnyedn felrhatunk egy foksorozatot, megfordtva azonban mr nem trivilis a problma, hiszen nem szksgszeren tartozik mint sorozathoz valban egy grf. Ezzel a krdssel sok helytt tallkozhatunk. Landau [14] az llatok kzt fennll dominancit modellezte ilyen mdon, Hakimi [4] kmiai, Ivnyi s munkatrsai pedig sportbeli [6, 7, 8, 10] alkalmazsokra hivatkoztak.

A problmra elsknt Vaclav Havel javasolt megoldst 1955-ben [5], majd tle fggetlenl 1962-ben Louis Hakimi [4] is publiklta a mdszert. Ebbl kifolylag

a mdszert rendszerint Havel-Hakimi algoritmusknt emlegetik. A mdszernek bizonyos szempontbl elnye, msfell pedig htrnya, hogy a sorozatokat nem csupn megvizsglja, hanem a hozzjuk tartoz grfot is ellltja, ezrt a legrosszabb futsi ideje nem lehet kisebb, mint ngyzetes.

Erds s Gallai 1960-ban [2] javasolt jobb mdszert a krds eldntsre, amely mr nem lltja el a grfot, csupn ellenriz. Az mdszerkhz ksbb Tripathy s Vijay [16] megadtak kt javaslatot amellyel az eredeti mdszer gyorsthat, de mg ezeket felhasznlva is nagyon komoly szmtsi ignnyel rendelkezik a feladat.

2011-ben megadtuk [8] az Erds-Gallai mdszer egy olyan vltozatt (EGL), amellyel lineris idben elvgezhet a vizsglat. A mdszert a ksbbiekben ismertetjk.

Sloan az interneten elrhet *Online Encyclopedia of Integer Sequences* [15] nev adatbzisban $n \leq 23$ cscs grfok esetn megtallhat, hogy pontosan hny klnbz foksorozat ltezik. Ezeknek a sorozatoknak a szmt a ksbbiekben γ_n-nel jelljk.

Az EGL algoritmust tovbb fejlesztve, a nagy mennyisg sorozat ellenrzsre ksztett EGE algoritmussal nagyobb n rtkekre is meghatroztuk a γ_n rtket. Ezeket az rtkeket egy tbb szmtgpen prhuzamosan futtatott hlzati program segtsgvel szmtottuk ki, mivel a szmtsok futsi ideje az n rtk nvelsvel minden lpsben krlbell ngyszeresre nvekszik [8]. Ez az exponencilis nvekeds tette szksgess az algoritmus szmtgpes megvalstsnak prhuzamostst.

Dolgozatunk clja a sorozatok gyors ellenrzsnek vizsglata s ismertetse, valamint a vizsglatot elvgz program bemutatsa.

A dolgozat felptse a kvetkez: a 2. rszben ismertetjk a Havel-Hakimi s az Erds-Gallai algoritmusokat, lerjuk Tripathy s Vijay gyorstsait, valamint a lineris Erds-Gallai algoritmust. A szmtsokat elvgz programot a 3. rszben ismertetjk, majd a 4. rszben ismertetjk a problma rszekre bontsnak technikit, vgl az 5. rszben sszefoglaljuk a lert mdszerekkel s programmal elrt eredmnyeinket.

2 Algoritmusok

A dolgozatban szerepl algoritmusok s forrskdjaik megtallhatak a (http://people.inf.elte.hu/lulsaai/H honlapon s a [8] cikkben.

Mieltt bevezetnnk a felhasznlt algoritmusokat, definiljuk, hogy pontosan mit is rtnk bizonyos fogalmakon s milyen alapvet felttelezsekkel lnk ezek kapcsn.

A G grf i-edik cscsa legyen V_i, V_i foka legyen f_i s foksorozata legyen $F = (f_1, \ldots, f_n)$.

Egy F foksorozatot *szablyos*, ha $n \geq 2$, az f_i, $2 < i \leq n$ rtkek egsz szmok s monoton cskkenek, azaz $f_2 \geq \ldots \geq f_n$. Egy szablyos foksorozatot *megvalsthatnak* vagy *helyrellthatnak* neveznk, ha ltezik olyan G egyszer grf, amelynek foksorozata F.

Szksgnk lesz tovbb a sorozat els k elemnek sszegre, amit $H_k = \sum_{i=1}^{k} f_i$ -vel fogjuk jellni.

2.1 Havel-Hakimi algoritmus

Az algoritmust elsknt Havel [5] rta le, majd ksbb Hakimi [4] tle fggetlenl publiklta ugyanezt az eredmnyt, innen addik a Havel-Hakimi algoritmus elnevezs.

(Havel, Hakimi [4, 5]) Az $F = (f_1, \ldots, f_n)$ foksorozat akkor s csak akkor helyrellthat, ha az $F' = (f_2-1, f_3-1, \ldots, f_{f_1}-1, f_{f_1+1}-1, f_{f_1+2}, \ldots, f_{n-1}, f_n)$ sorozat is helyrellthat.

Lsd [4, 5].

Az algoritmust ksbb kiterjesztettk irnytott grfokra is [3, 12, 13] s ltalnostottk irnytott multigrfokra [6, 7].

A ttel felhasznlsval felrhat algoritmus nagy htrnya, hogy felhasznlsa sorn minden egyes lpsben jra kell rendezni a sorozatot. Ezrt az algoritmus futsi ideje legrosszabb esetben akkor is $\Omega(n^2)$, ha lineris futsi idej rendez algoritmust hasznlunk.

2.2 Eltol Havel-Hakimi algoritmus

Az ellenrizend sorozat egyre kisebb s kisebb rsznek rendezst megvalsthatjuk gy is, hogy csupn azokat az elemeket toljuk el a sorozatban, amelyekre nem teljesl a monotonits. Ezzel a rendezshez szksges id nagy rszt megtakarthatjuk. Ez az algoritmus az eltol Havel-Hakimi algoritmus.

2.3 Erds-Gallai algoritmus

Erds s Gallai 1960-ban adtak meg olyan ellenrzsi mdot, amely a sorozatot mr nem lltja el, csupn eldnti, hogy az helyrellthat vagy sem. *(Erds, Gallai, [2])* A szablyos $F = (f_1, \ldots, f_n)$ sorozat akkor s csak akkor helyrellthat, ha igaz az, hogy

$$\sum_{i=1}^{n} f_i \quad \textit{páros} \tag{1}$$

s

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

Lsd [1, 2].

A ttel felhasznlsval megvalsthat algoritmus futsi ideje a legjobb $\Theta(n)$ s legrosszabb $\Theta(n^2)$ kztt vltozik.

A kzelmltban Tripathi s munkatrsai [17] publikltak egy konstruktv bizonytst, amely helyrellthat sorozatok esetn a helyrelltst is elvgzi. Ennek az algoritmusnak a futsi ideje legrosszabb esetben $\Theta(n^3)$.

2.4 Rvidtett Erds-Gallai algoritmus

Tripathi 2003-as cikkben [16] ler egy lehetsges rvidtsi mdot Erds s Gallai mdszerre: A (2) egyenltlensget elegend csak azokban az esetekben ellenrizni, ahol a $H_i > i(i-1)$ felttel teljesl.

(*Tripathi et al. [16]*) Egy szablyos $F = (f_1, \ldots, f_n)$ foksorozat akkor s csak akkor j, ha

$$H_n \quad p\acute{a}ros \tag{3}$$

s

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

ahol

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

Lsd [16].

2.5 Ugr Erds-Gallai algoritmus

Tripathi s Vijay [16] megadtak egy tovbbi lehetsges rvidtsi mdot is. A rvidts azt hasznlja ki, hogy ha egy ellenrizend sorozatban egy fokszm tbbszr is elfordul, akkor elegend azt csupn egyszer ellenrizni, mghozz abban a pontban, ahol teljesl, hogy $f_i \neq f_{i+1}$. A sorozatok utols elemeit pedig nem szksges ellenrizni.

Vezessk be a σ_i jellst azokra az i indexekre, amikre teljeslnek az $f_i \neq f_{i+1}$ s $i \neq n$ felttelek. Nevezzk ezeket a pontokat *ugrpontoknak*. Ezzel a jellssel teht a sorozat $f_{\sigma_1}, \ldots, f_{\sigma_q}$ elemei ugrelemek s q jelli a sorozat ugrelemeinek szmt.

(Tripathy, Vijay [16]) A szablyos $F = (f_1, \ldots, f_n)$ sorozat akkor s csak akkor helyrellthat, ha igaz az, hogy

$$H_n \quad p\acute{a}ros \tag{6}$$

s

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

Lsd [16].

2.6 Lineris Erds-Gallai algoritmus

Az algoritmust tovbb gyorsthatjuk, ha kihasznljuk az f sorozat monotonitst, azaz azt, hogy a felttelben szerepl sszegzst nem kell minden egyes iterciban elvgezni, mivel a szummban szerepl min fggvny egy τ_i indexig mindig az els, majd attl kezdve mindig a msodik paramtert fogja visszaadni. A monotonits

miatt pedig ez az rtk csak az egyik irnyban vltozhat, gy elg az egyes itercik kztt nhny lpsben eltolni, gy nyernk egy explicit kpletet a szummra, amit korbban mindig jra s jra kiszmtottunk.

A szablyos $F = (f_1, \ldots, f_n)$ sorozat akkor s csak akkor helyrellthat, ha igaz az, hogy

$$H_n \quad \textit{páros} \tag{8}$$

s

$$H_{\sigma_i} < H_n - H_{\tau_i} + \sigma_i \cdot (\tau_i - 1) \tag{9}$$

Az ellenrzpontok elgsgessgt Tripathi s munkatrsai [16] mr bebizonytottk. A ttelben megadott felttel ezeket az ellenrzseket vgzi el, kihasznlva a sorozat elemeinek monoton cskkenst, azaz a

$$\sum_{k=\sigma_i+1}^{n-1} \min(i, f_k) \tag{10}$$

tagot nem szmolja jra minden esetben, pontosabban nem ebben a formban vgzi el a szmtst, hanem explicit mdon.

A kifejezs rtke a (11) formban adhat meg, mgpedig azrt, mert a sorozat monotonitsa garantlja, hogy a τ_i slypont eltt a $\min(i, f_k)$ kifejezs rtke minden esetben i, majd a τ_i-nl nagyobb vagy egyenl k-k esetben mindvgig f_k. Ebbl kvetkezik, hogy

$$\sum_{k=\sigma_i+1}^{n-1} \min(i, f_n) = H_n - H_{\tau_i} + \sigma_i \cdot (\tau_i - \sigma_i). \tag{11}$$

Az eddigiek alapjn az eredeti felttelt trhatjuk a kvetkez alakba

$$H_{\sigma_i} - \sigma_i \cdot (\sigma_i - 1) \leq H_n + H_{\tau_i} + \sigma_i \cdot (\tau_i - \sigma_i). \tag{12}$$

A (13) egyenltlensget trendezve megkapjuk a

$$H_{\sigma_i} \leq H_n + H_{\tau_i} + \sigma_i \cdot (\tau_i - 1). \tag{13}$$

kifejezst, ami pontosan egyezik a (9) egyenltlensggel.

A mdszer linearitsa abbl fakad, hogy a tesztelshez a parits ellenrzssel egytt kt felttelvizsglat, kt kivons s egy sszeads szksges. A τ_i paramter meghatrozsa egy lineris keresssel elvgezhet – amelyhez esetnkben legrosszabb esetben $2 \cdot n$ mvelet szksges, ami tartalmazza az sszes τ_i rtk meghatrozst az adott sorozathoz.

2.7 Erds-Gallai-Enumerate

Az fent ismertetett lineris idej algoritmust felhasznlva megadhat egy olyan algoritmus, amely lexikografikus sorrendben ellltja az sszes nullmentes, pros sorozatot s ellenrzi azokat. Ez az algoritmus kpezi a 3.2 rszben ismertetend program alapjt.

Bemenet. n: a cscsok szma ($n \geq 1$);
b: az els rszfeladat esetben a kezdsorozat, egybknt az aktulis rszfeladatot megelz feladat utols ellenrztt sorozata;
$lastIndex$: az utols sorozat azonostsra hasznlt index;
$lastValue$: az utols sorozat $lastIndex$. elemnek rtke.

Kimenet. $\zeta_{n,m}$: a ζ_n rtk m. rszletsszege.

ERDOS-GALLAI-ENUMERATE$(n, b, lastIndex, lastValue, \zeta_{n,m})$

```
01 if b_n ≠ n − 1                                  Line 01: not the first sequence
02       b_n = b_n − 3                             Lines 02–03: decrease last element by 3
03       H_n = H_n − 3
04 else                                            Line 04: first sequence
05       b_n = b_n − 2                             Lines 05–06: decrease last element by 2
06       H_n = H_n − 2
07 end
08 for i = 1 to n
09       J_i = n − 1                               Lines 08–10: fill up J with n − 1
10 end
11 H_1 = b_1                                       Lines 11–14: calculate H vector
12 for i = 2 to n
13       H_i = H_{i−1} + b_i
14 end
15 c = 0                                           Line 15: set number of checkpoints to 0
16 for i = 1 to n − 2                              Lines 16–21: fill up checkpoint vector
17       if b_i ≠ b_{i+1} and b_i ≠ f_n
18           c = c + 1
19           D_c = i
20       end
21 end
22 ζ = Erdos − Gallai − Check(b, H, c, D)          Line 22: check first job
23 while b_{lastIndex} > lastValue                 Line 23: till the last sequence of job
24       k = n
25       if b_k ≥ 3                                Line 25: if last element in sequence greater than 3
26           b_k = b_k − 2                         Line 26: decrease last element by 2
27           H_k = H_k − 2
28           if c < 1 or D_c ≠ n − 1               Line 28: if no checkpoints yet
```

```
29          c = c + 1                          Line 29–30: add new checkpoint
30          D_c = n − 1
31      end
32  else
33      if b_k = 2                  Line 33: if last element in sequence equals 2
34          b_{k−1} = b_{k−1} − 1   Line 34: decrease penultimate element by 1
35          H_{k−1} = Hk − 1 − 1
36          if (c = 1 and D_c ≠ n − 2) or          Line 36: n − 2 not a c.p.
                   (c > 1 and D_{c−1} ≠ n − 2 and D_c ≠ n − 2)
37              if c > 0 and D_c > n − 2   Lines 37–44: set n − 2 as c.p.
38                  D_{c+1} = D_c
39                  D_c = n − 2
40              else
41                  c = c + 1
42                  D_c = n − 2
43              end
44          end
45          if b_{n−1} odd
46              b_n = b_{n−1}       Lines 46–49: fix sequence and checkpoints
47              if c > 0 and D_c = n − 1
48                  c = c − 1
49              end
50          end
51      else                                          Line 51: b_n even
52          b_n = b_{n−1} − 1   Line 52–56: fix last element and checkpoints
53          if c > 0 and D_c ≠ n − 1
54              c = c + 1
55              D_c = n − 1
56          end
57      end
58      H_n = H_{n−1} + b_n
59      end
60  else                                          Line 60: last element is 1
61      j = n − 1                   Line 61–64: find last element that ≠ 1
62      while b_j ≤ 1
63          j = j − 1
64      end
65      if b_j ≥ 3
66          b_j = b_j − 1
67          H_j = H_j − 1
68          if j > 1                          Line 68: b_j is not the first element
69              if (c = 1 and D_c ≠ j − 1) or   Line 69–78: refresh c.p.s
                       (c > 1 and D_{c−1} ≠ j − 1)
70                  if c > 0 and D_c > j − 1
```

```
71                      D_{c+1} = D_c
72                      D_c = j − 1
73                      c = c + 1
74                  else
75                      c = c + 1
76                      D_c = j − 1
77                  end
78              end
79          end
80          for k = j + 1 to n                Line 80–83: refresh the sequence
81              b_k = b_j
82              H_k = H_{k−1} + b_k
83          end
84          while c > 1 and D_c > j − 1       Line 84–86: refresh checkpoints
85              c = c − 1
86          end
87          if H_n odd                        Line 87: if b sequence is broken
88              b_n = b_n − 1                     Line 88–89: fix last element
89              H_n = H_n − 1
90              if D_c ≠ n − 1                    Line 90–93: fix checkpoints
91                  c = c + 1
92                  D_c = n − 1
93              end
94          end
95      else                                  Line 95: b_j < 3
96          b_{j−1} = b_{j−1} − 1
97          H_{j−1} = H_{j−1} − 1
98          if j > 2
99              if (c ≤ 2 or                  Lines 99–113: fix checkpoints
                (c > 1 and D_{c−1} ≠ j − 2)) and (c > 1 and D_{c−2} ≠ j − 2)
100                 if c > 1 and D_{c−1} > j − 2
101                     D_{c+1} = D_c
102                     D_c = D_{c−1}
103                     c = c + 1
104                 else
105                     if c > 0 and D_c > j − 2
106                         D_{c+1} = D_c
107                         D_c = j − 2
108                         c = c + 1
109                     else
110                         c = c + 1
111                         D_c = j − 2
112                     end
113                 end
```

```
114             end
115         end
116         for h = j to n                          Lines 116–119: fix sequence
117             b_k = b_{j-1}
118             H_n = H_{k-1} + b_k
119         end
120         while c > 1 and D_c > j - 1   Lines 120–122: fix checkpoints
121             c = c - 1
122         end
123         if H_n odd                              Line 123: sequence is broken
124             b_n = b_{n-1} - 1                   Lines 99–113: fix last element
125             H_n = H_{n-1} + b_n
126             if D_c ≠ n - 1                      Lines 126–129: fix checkpoints
127                 c = c + 1
128                 D_c = n - 1
129             end
130         end
131     end
132 ζ = ζ + Erdos - Gallai - Check(b, H, c, D)
                                    132. line: check if sequence graphical
133 end
```

Ez az algoritmus lehetv teszi, hogy egyszerre prhuzamosan tbb gppel vgezzk a ζ rtkek kiszmtst. Mghozz gy, hogy a kezdsorozattl az utols sorozatig minden sorozatot letesztel. Az utols sorozatot – a kezd sorozattal ellenttben – nem adjuk meg bemenetknt, csupn az azonostshoz hasznlt kt rtket: $lastIndex$, $lastValue$. Ezekkel az rtkekkel a kvetkez mdon azonosthat a sorozat. Mivel a sorozatokat lexikografikus sorrendben lltjuk el, ezrt a sorozat csupn egyetlen tagjnak ellenrzsvel ($f_{lastIndex}$) eldnthet, hogy elrtk-e mr az utols ellenrizend sorozatot, azaz teljesl-e, hogy $f_{lastIndex} < lastValue$. Ha igen, akkor a kiosztott feladatot elvgzettnek tekinthetjk.

3 A Holzhacker program

A γ_n rtkek meghatrozsa komoly erforrsignnyel br, ezrt szmtsainkat az albbi technikk felhasznlsval igyekeztnk felgyorstani:

- a feladat rszekre bontsa s sztosztsa tbb szmtgp kztt,
- a nyilvnvalan rossz sorozatok elhagysa,
- csak nullmentes sorozatok vizsglata,
- ellenrzs elvgzse csak ugrpontokban,

- a 2.5 s 2.6 pontokban lert rvidtsek hasznostsa,

- annak kihasznlsa, hogy az ugrpontok listja konstans id alatt frissthet, ha a vizsgland nullmentes, pros sorozatokat lexikografikus sorrendben lltjuk el.

A feladat prhuzamostsa a processzorok szmval arnyosan cskkenti a szksges futsi idt, azaz minl tbb szmtgppel dolgozunk, annl kevesebb ideig tartanak a szmtsok.

Nyilvnvalan rossz sorozatoknak tekintjk azokat a sorozatokat, amelyekben a fokszmok sszege pratlan.

A nulls sorozatok vizsglata pedig azrt hasznos, mert a sorozatok megkzeltleg negyed rszben szerepel nulla. Mivel minden egyszer grfot reprezentl foksorozatra teljesl, hogy egy nullt, azaz egy izollt cscsot hozzvve a kapott sorozat tovbbra is egy grfot reprezentl, ezrt ezeket a sorokat felesleges megvizsglnunk, mert γ_n szmtsa sorn jra leellenriznnk minden γ_{n-1} szmtsa sorn korbban mr megvizsglt sorozatot. Ezt az tletet hasznlja fel a kvetkez lemma.

Az n-cscs grfok szma (γ_n) megadhat az $n-1$ pont megvalsthat grfsorozatok s az n hossz j nullmentes sorozatok sszegeknt, azaz

$$\gamma_n = \zeta_n + \gamma_{n-1},$$

ahol ζ_n a helyrellthat n-hossz nullmentes sorozatok szma.

Lsd [8].

A program mkdsnek lersa sorn dlt kisbetk jellik a felhasznlt vltozkat s dlt nagybetk a konstansokat, valamint a fontosabb hlzati primitveket.

3.1 Kiszolgl program

Programunkban egy kzponti kiszolgl program osztja ki a rszfeladatokat a szmtsokhoz csatlakoz kliensek kztt. Ennek ksztsekor az elsdleges szempontunk a lehet legnagyobb megbzhatsg, rugalmassg s sebessg elrse volt.

Megbzhatsgon azt rtjk, hogy a rendszert mkdst kls termszeti vagy akr humn tnyezk ne befolysoljk. A kiszolgl program ezrt brmikor lellthat s jraindthat anlkl, hogy ez eredmnyek elvesztsvel jrna. A szerver lellsa a klienseket sem befolysolja. Ez egyben azt is jelenti, hogy amennyiben szksges, a szerver programot egy komolyabb szmtsi projekt sorn brmikor kicserlhetjk vagy megvltoztathatjuk.

A szerver rugalmassga abban rejlik, hogy mkdse semmilyen mdon nem fgg ssze azzal, hogy a kliensekben mi trtnik, csupn annyi a teendje, hogy feladatokat oszt ki s eredmnyeket gyjt be. Az sem jelent problmt, ha kzben minden klienst megllt unk, mert ezzel a szerver egyltaln nem foglalkozik. Ez elsre ugyan rdektelennek tnhet, de nem az, ha figyelembe vesszk, hogy a kliens brki szmra szabadon elrhet az interneten, gy szmtani kell arra, hogy valaki gy dnt, hogy rszt vesz egy komoly szmtsban s elindtja a klienst, majd

meggondolja magt s mgis inkbb lelltja azt. Ehelyett gondolhatunk akr egy egyszer ramsznetre is. Tovbb lpve, mivel nincs llad hlzati kapcsolat a kiszolgl s a tbbi gp kztt, gy nincs limitlva a rsztvev szmtgpek szma sem. Ha valaki elindt a szmtgpn egy klienst, az kap egy feladatot s szmol.

Harmadik szempontunk a sebessg volt, ezrt programunkat C nyelven valstottuk meg. Vlasztsunk azrt erre a nyelvre esett a kezdeti MATLAB programok helyett, mert a MATLAB ugyan egy nagyon kellemes s jl tlthat futsi felletet biztost, de mrseink alapjn C-vel megkzeltleg szzszoros gyorsuls rhet el. Programunk teht C-ben kszlt, Berkeley Socketek felhasznlsval. Mkdse pedig a kvetkez:

[scale=0.6]pic/server

Figure 1: A szerver mkdsnek folyamatbrja

- A szerver indtsakor ltrehoz egy csatlakozt (*SOCKET*), amit aztn egy csatlakozsi ponthoz (*PORT*[1]) kt (*BIND*). Amint ltrejtt a csatlakozsi pont s a szerver alkalmas a kapcsolatok fogadsra, ellenrzi, hogy van-e ksz eredmny a naplfjlban. Amennyiben a napl nem res, gy az abban szerepl feladatokat ksznek jelli. Ezutn elkezddhet a kliensek bejv kapcsolatainak kezelse (*LISTEN*).

- Mindaddig, amg a befejezett feladatok szma (*finished_jobs*) el nem ri az sszes feladat szmt (*NUM_JOB*), addig jra s jra klienseket fogadunk, majd a fogadott kliens zenetbl eldntjk, hogy mi clbl jtt. A kliens kt okbl csatlakozhatott: j feladatot kr vagy elkszlt egy rszeredmnnyel s azt szeretn visszakldeni.

[scale=0.6]pic/newjob

Figure 2: j feladat kiosztsa a kliensnek

- Ha a kliens j feladatot kr, akkor a 2. brn lthat mdon kiosztjuk neki a soron kvetkezt a kvetkez algoritmus alapjn:

  + Megnveljk az ppen aktulis rszfeladat sorszmt trol rtket (*actual_job*), majd ellenrizzk, hogy nem lptnk-e tl a lehetsges legnagyobb sorszmon (*NUM_JOB*), valamint hogy a feladatok sttuszt trol vektorunk (*job_status*) alapjn nincs-e mr teljestve a megadott feladat.
  + Addig ismtelgetjk ezt a mveletet, amg meg nem talltuk a kiosztand feladatot.
  + Miutn a feladat kikldsre kerlt, lebontjuk a kapcsolatot. Itt megjegyezzk, hogy ha elrtk az utols feladatot, akkor mdszeresen elkezdjk jra kiosztani a mr legrgebben kiadott feladatokat arra az

[1] A programban elre definilt rtk.

esetre, ha a kliens, amely azt a feladatot vgezte volna, valamilyen okbl (szmtgp lelltsa, ramsznet stb.) nem tudta azt befejezni. Ez nyilvn azt eredmnyezi, hogy amikor mr nagyon kevs munka van vissza, akkor sok kliens fogja ugyanazt a feladatot megkapni, gy a feladatok befejezdsvel egyre robusztusabb vlik a rendszer.

[scale=0.6]pic/gotresult

Figure 3: Rszfeladat eredmnynek mentse

- Ha a kliens eredmnyt kld, akkor a 3. brn lthat algoritmus szerint a kvetkez mdon dolgozzuk azt fel:

  + Fogadjuk az zenett s ellenrizzk, hogy a berkez eredmny nem szerepel-e mr a teljestett feladatok listjban (*job_status*); amennyiben nem szerepel, gy elmentjk s befejezettnek tekintjk.

  + A ments sorn a kliens ltal kldtt zenet eredeti formban kerl hozzfzsre a napl fjlhoz (*LOG_FILE*). A napl fjlban csv formtumban kerlnek rgztsre az adatok. A fjl pontos formtumt a 3.2. rszben ismertetjk.

3.2 Kliens program

A kliens program elksztsekor a f cl a lehet legnagyobb egyszersg elrse volt. Olyan programot akartunk kszteni, amely nem ignyel semmifle felhasznli beavatkozst, belltst vagy akrmi mst az elindtsn kvl. Ez azrt fontos, mert a szmtsokat a lehet legtbb szmtgp kztt szeretnnk sztosztani, tekintettel az risi futsi idkre.

Fontos volt az is, hogy ha egy komolyabb szmts befejezdik, akkor se kelljen minden egyes gpen jra s jra elindtani a programot, hanem az a lehet legkisebb erforrs ignnyel maradjon a httrben s vrjon mindaddig, amg rszt nem vehet egy jabb szmtsban. Ez sokat segtett, amikor az j γ rtk szmtst vgeztk tbb mint 200 szmtgp felhasznlsval. Ezek kztt a gpek kztt voltak olyanok, ahol a processzorban ngy mag is volt, gy egyszerre ngy klienst kellett volna jraindtani, amikor az egyik γ rtk kiszmtst kveten ttrtnk egy nagyobbra.

Jogosan felmerlhet a krds, hogy mirt nem elegend egyetlen kliens futtatsa tbb szlon gpenknt. A szmtsokat nem csak laborgpekkel vgeztk, hanem programunkat publikusan elrhetv tettk az interneten, gy brki csatlakozhatott, ezrt nem szerettk volna, hogy valaki, aki egy klienst futtatva a gpn segthetn munknkat, azrt ne hasznlja azt, mert az teljesen leterheli a szmtgpt.

A kliens program mkdse a kvetkez:

- Miutn ltrehoztuk a csatlakozshoz szksges csatlakozt (*SOCKET*), megprblunk csatlakozni a kiszolgl programhoz. Ha ez nem lehetsges, mert

feladat sorszma	els sorozat		
	utols sorozat	futsi id	j sorozatok szma

Table 1: A kliens ltal kldtt eredmnyek szerkezete

a szerver nem elrhet, akkor *wait_time*[2] ideig vrakozunk s egyttal megkt-szerezzk a vrakozsi idt. Ezt addig ismtelgetjk, amg fel nem pl a kapcsolat vagy el nem rjk a *MAX_WAIT_TIME* rtket. Ezt kveten mr nem nveljk tovbb a vrakozs idtartamt. A mdszer alaptlett Jacobson 1988-as [11], lass kezdst biztost protokollja adta. Megjegyezzk, hogy a neve ellenre a mdszer egyltaln nem lass, hiszen exponencilis nvekedst eredmnyez.

- A kapcsolat ltrejttt kveten zenetet kldnk a szerver programnak, amellyel egy feladatot krnk tle, majd miutn megkaptuk azt, lebontjuk a hlzati kapcsolatot s a 2.7. rszben ismertetett algoritmus segtsgvel ellltjuk s ellenrizzk a kiosztott feladatba tartoz sorozatokat.

- A kliens az ellenrzsek sorn csak olyan sorozatokat llt el s ellenriz, amelyek prosak s nullmentesek. Ez all egyetlen kivtel lehetsges: pratlan n rtkek esetn az utols ellenrztt sor nem az $1, 1, \ldots, 1$, hanem a $-1, -1, \ldots, -1$, mivel a sorozatokat generl algoritmus kettesvel lpteti a sorozatok tagjait anlkl, hogy 0 elfordulna, gy az utols eltti ellenrztt sorozat a $2, 1, 1, \ldots, 1$ mely a kvetkez lpsben – mivel nulla nem lehet – $-1, -1, \ldots, -1$-re vlt. Ez azonban nem jelent problmt, mert az ellenrz algoritmus elutastja.

- A szmtsok vgeztvel a kapott eredmnyeket elkldjk a szervernek, amely aztn azokat a 1. tblzatban lthat formtumba elmenti. A tblzat res celli ksbbi felhasznlsra vannak fenntartva.

- A kiszmtott eredmnyek elkldse is hasonl mdon trtnik, mint ahogy a feladatot krtk a szervertl. Azonban most nem prblkozunk a vgtelensgig. sszesen *MAX_RECONNECT_TRY* alkalmat kveten abbahagyjuk a prblkozst. Ez azrt fontos, mert ha a szerver megkapott minden rszeredmnyt lell, de a kliensek tovbb dolgoznak. Ezzel az elsdleges clunk, hogy ne kelljen a klienseinket minden egyes projekt esetn jra s jra elindtani mindentt, hanem ha a szerver huzamosabb ideig nem rhet el, akkor dobjk el rszeredmnyeiket s folytassk a normlis mkdst, azaz krjenek j feladatot, pontosabban vrakozzanak j feladatra. Ezzel ismt csak a rendszer hasznlatnak egyszerstst s nagyobb hibatrst akartunk elrni.

[2]Elre definilt rtk. Alaprtelmezse *DEFAULT_WAIT_TIME*.

4 A problma felosztsa

A programok mkdst korbban mr ismertettk. Most lerjuk azt is, hogy a problmt milyen technikk segtsgvel osztottuk fel a szmtgpek kztt. Elsjnt lerjuk a legegyszerbb naiv megkzeltst, amely kisebb n rtkek esetn mg megfelel, majd ttrnk arra, hogy nagyobb n-eknl milyen mdszerekkel prbltuk kiegyenslyozni az egyes szmtgpek terhelst.

4.1 Nav megkzelts

Legels megkzeltsnk szerint a problmt kisebb n rtkek esetn tapasztalt futsi idk alapjn igyekeztnk azonos mret darabokra osztani. Ez γ_{24} esetn mg elfogadhat eltrseket eredmnyezett, de sajnos nagyobb n-ekre mr olyan mrtk lett volna az eltrs, ami rontotta volna a prhuzamosts hatkonysgt.

Ezt a megkzeltst felhasznlva a problmt sszesen 23 rszproblmra bontottuk. A futsi idkre vonatkoz adataink kzlstl eltekintnk, tekintve, hogy a rszfeladatok tbb egymstl eltr hardverrel felszerelt szmtgpen futottak.

4.2 Feloszts korbbi n rtkekre alapozva

Az elz megkzelts legnagyobb hinyossga a rendszertelensg volt, hiszen se az sszesen megvizsglt sorozatok szmt, sem az elfogadott sorozatokat nem vette szmtsba. Az elfogadott sorozatok szmt fontos tnyezknt kell kezelni, hiszen ezekben az esetekben a teszt minden egyes ellenrz pontban lefut, gy ezek a tesztek veszik ignybe a legtbb idt.

A sorozatok szerinti felosztst a kvetkez mdon generltuk. A korbban sorozatok generlsra felhasznlt programunkat gy alaktottuk t, hogy a tesztet eltvoltottuk, csupn azt figyeltk, hogy hny sorozaton vagyunk tl. Mivel az sszes s a pros sorozatok szmt ismertk, gy knnyedn meghatrozhat volt, hogy mikor vagyunk kt lehetsges rszfeladat hatrn. A kapott sorozatot teljes egszben trolni azonban nem lett volna clszer, hiszen az rendkvl nagy mrtkben lasstan a szmolst ha minden egyes sorozat ellenrzst kveten ellenriznnk kellene az ppen aktulis sorozat minden elemt, hogy nem rtk-e el az utols ellenrizend sort. A kapott sorozatokrl teht csak annyi informcit troltunk, hogy mi az els elem indexe, ami eltr az azt megelz sorozattl, s hogy mi ez az rtk. Ezeket az informcikat felhasznlva mr elegend egyetlen ellenrzst elvgezni, hogy eldnthessk, vgeztnk-e a kapott feladattal.

Ez a „csonkols” azonban jra csak azt eredmnyezi, hogy a rszfeladatok eltr mretek lesznek. Ez a problma aztn egyre nagyobb eltrseket okoz, ha az n rtk nvekszik.

Ezzel a technikval egy 435 rszfeladatot tartalmaz felosztst ksztettnk s hasznltunk fel a ζ_{25} s ζ_{26} rtkek szmtshoz. Ezekben az esetekben mr nem a γ rtkeket szmtottuk. Ennek oka, hogy a 3. lemma alapjn γ rtkt knnyedn

meghatrozhatjuk ζ ismeretben s gy a szksges szmolsi id az eredeti $\frac{3}{4}$-re cskken [8].

4.3 Feloszts a mveletszmok alapjn

Az elz megoldsunk mr viszonylag j eloszlst biztostott a futsi idk tekintetben, azonban elfordul benne $2-3$ ris feladat is, amelyek megoldsa akr tzszer annyi ideig is tartott, mint a tbbiek. Ezt a jelensget az albbi mdon prbltuk meg kikszblni.

A generlt rszsorozatokat mr nem futsi idejk vagy az ellenrztt sorozatok szma alapjn osztottuk fel, hanem az ellenrzsek sorn elvgzett mveletek szma adta a sorozatok kzti hatrokat. Ezt gy hajtottuk vgre, hogy egy alacsonyabb n rtkre kiszmtottuk a teljes mveletignyt, majd ennek ismeretben generltuk az j feladatokat. Nyilvn ennek a felosztsnak is velejrja volt, hogy sorozatainkat „csonkoltuk" a nagyobb sebessg elrsnek rdekben.

A ζ_{28} rtkt ezt a felosztst felhasznlva szmtottuk ki.

Feloszts a sorozatok tnyleges szmnak kzeltse alapjn

A korbbi megvalstsok ugyan kzeltleg megfelel felosztst eredmnyeztek, azonban mindegyikben fordultak el – mg ha csak kis szmban is – mamutok[3]. Ennek elkerlsben volt hasznos a most kvetkez lemma.

Az n hossz sorozatok szma, melyek p-edik eleme m

$$N(n, m, p) = \binom{m + (p-1)}{n} \tag{14}$$

A lemmt felhasznlva megadhat egy mtrix, amely megadja, hogy kzeltleg hny olyan sorozat van, amely a b_1 rtkkel kezddik s b_2 rtkkel folytatdik stb. Ezt a tblzatot felhasznlva a kvetkez mdon adhat meg a feloszts:

- Vlasszuk meg a maximlis feladat mretet (ez esetnkben akkora szeleteket jelentett, amelyek egy jszaka alatt kiszmthatak).
- Induljunk el a mtrix als sornak els elemtl s kezdjk a mtrix tovbbi sorait kiolvasni s sszegezni mindaddig, amg az aktulis kapacits vagy a kiolvasand rtkek el nem fogytak.
- Ha egy rtk tl nagy az adott maximlis mrethez viszonytva, akkor lpjnk egy sorral feljebb s kezdjk el azt a sort kiolvasni addig az oszlopig, ahonnan fellptnk.
- Folytassuk ezt az algoritmust, amg az utols sor vgig nem rtnk.

A korbban ismertetett algoritmus szerint $n = 5$-re a 2. tblzatbl kiolvashat a 3. tblzatban lthat feloszts.

[3] Az tlagos futsi id tbbszrst ignyl rszfeladat.

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

Table 2: A felosztshoz hasznlt mtrix $n = 5$ esetn

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

Table 3: $n = 5$-s feloszts hatrsorozatai

A tblzatban lthat sorozatok a kvetkez mdon rtendek: az els szelet a $1, 1, 1, 1, 1$ sorozattl a $2, 2, 2, 2, 2$ sorozatig tart, a msodik pedig a $2, 2, 2, 2, 2$ sorozattl a $3, 2, 2, 2, 2$ sorozatig s gy tovbb.

Ez a mdszer ugyan nem garantlja, hogy a rszfeladatok hossza pontosan ugyanakkora legyen, azonban megszlteti a korbbiakban tapasztalt „mamut" hatst.

Ezt a mdszert hasznltuk fel ζ_{29} kiszmtsakor. A teljes szmtst sszesen tbb mint 15.000 rszfeladatra felosztva vgeztk.

A teljes feloszts generlsnak els lpse a 2. tblzathoz hasonl mtrix elllltsa a megfelel n rtkhez. A mtrix feltltst vgzi el az albbi program.

Bemenet. n: a cscsok szma ($n \geq 1$);
$MaxSize$: a maximlis megengedett „szeletmret" (maximlis sorozatszm a szeletben).

Kimenet. a kpernyre rjuk a szeleteket hatrol sorozatokat.

GenMatrix$(n, MaxSize)$

```
01 for i = n downto 2                    01–05. sor: a mtrix feltltse
02     for j = 1 to n − 1
03         M_{i,j} = C(i+j−2, i−1)
04     end
05 end
06 for j = n − 1 downto 1                06–08. sor: a mtrix els sornak kitltse
07     M_{1,j} = 1
08 end
```

09 **GenSeries**$(M, n, n, 1, n-1, MaxSize, 0)$ 09. sor: szelet geners

Miutn megvan a mtrixunk, mr csak ki kell olvasnunk belle a szeletek hatrsorozatait (a 3. tblzathoz hasonl alakban) az albbi algoritmus segtsgvel.

Bemenet. M: a felosztshoz tartoz mtrix, amit a $GenMatrix$ algoritmussal kaptunk;
n: a cscsok szma ($n \geq 1$);
i, j, j_m, J: rekurzv segdparamterek;
$MaxSize$: a maximlis megengedett szeletmret (maximlis sorozatszm a szeletben).

Kimenet. A szeleteket hatrol sorozatok (rekurzv).

GenSeries$(M, n, i, j, jm, MaxSize, J)$

```
01 C = 0                                          01. sor: a szelet mretnek inicializlsa
02 while j < j_m + 1
03     if C + M_{i,j} ≤ MaxSize                   03. sor: ha bvthet a szelet
04         C = C + M_{i,j}                        04. sor: bvts
05         if j ≤ j_m                             05–07. sor: tovbb lpnk a mtrixban
06             j = j + 1
07         end
08     else                                       08. sor: nem bvthet a szelet
09         if C ≠ 0                               09. sor: a szelet nem res
10             for k = 2 to size(J, 2)            10–16. sor: kirs
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: sortrs
17             C = 0
18         end
19         if M_{i,j} > MaxSize and j ≤ j_m       19. sor: felbonthatsg ellenrzse
20             GenSeries(M, n, i − 1, 1, j, MaxSize, [J, j])
21             j = j + 1
22         end
23     end
24 end
25 if C ≠ 0                                       25. sor: az utols szelet nem res
26     for k = 2 to size(J, 2)                    26–32. sor: utols szelet kirsa
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
```

5 Eredmnyek

A korbban ismertetett Holzhacker program segtsgvel bvtettk az *On-line Encyclopedia of Integer Sequences* [15] adatbzist, ami γ_{23}-ig tartalmazta a grfsorozatok szmt. Programunk segtsgvel megadtuk a hinyz rtkeket egszen γ_{29}-ig. Ezek az rtkek megtallhatak a 4. tblzatban.

Szmtsaink sorn ignybe vettnk laborgpeket (komputeralgebra labor, nyelvi labor, adatbzis labor, lovarda) valamint magnszemlyek segtsgt is. sszegezve a szmtgpek szmtsi teljestmnyt, azt mondhatjuk, hogy szmtsi kapacitsunk elmleti maximuma elrte a 6 TFLOPS sebessget. Rszletesebb adatok tallhatak a 5. tblzatban.

A szmtsok sorn ltrejtt naplfjlok a forrskdokkal egytt megtallhatak a honlapomon (http://people.inf.elte.hu/lulsaai/Holzhacker).

Az ltalunk kiszmtott γ rtkek napli kzl a legrvidebbet tartalmazza a 6. tblzat rvidtett formban, azaz nem tartalmazza a szeletek kezd s befejez sorozatait, csak a sorszmot, futsi idt s az elfogadott sorozatok szmt. Itt megjegyezzk, hogy ebben a szmtsban mg nem vettek rszt a laborok gpei.

A klnbz ζ rtkek rszfeladatainak futsi idt sszegezve megkaphatjuk, hogy a teljes szmts egy gppel mennyi idt vette volna ignybe. Ezeket az adatokat tartalmazza a 7. tblzat.

6 sszefoglals

A dolgozat tmja grfok foksorozatainak gyors tesztelsre volt, amelynek sorn klasszikus mdszerek felhasznlsa mellett megadtunk egy j, lineris futsi idej mdszert, bemutattunk egy foksorozatokat megszmll programot s az azzal elrt eredmnyeinket.

Az els rszben rviden bemutattuk az alap problmt, annak alkalmazsi terleteit s a dolgozatban elrni kvnt clokat.

A msodik rszben ismertettk az eredeti Erds-Gallai mdszert s a Tripathy s munkatrsai ltal megadott gyorstsokat. Ezutn lertuk a lineris Erds-Gallai, valamint az Erds-Gallai-Enumerate algoritmust, amelyeket a ksbbiekben a grfsorozatokat megszmll programban hasznltunk fel.

A dolgozat harmadik rszben adtuk meg a Holzhacker program ismertetst, kitrve a programm fbb sajtossgaira, valamint a megvalstst meghatroz fbb irnyelvekre s clokra.

A negyedik rsz tartalmazta a grfsorozatok felosztsnak megvalstsra felhasznlt mdszereket s algoritmust, amiket felhasznltunk az j γ s ζ rtkek

n	ρ_n	ϵ_n	γ_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**

Table 4: sszes, pros s helyrellthat sorozatok szma.

Labor / Tulajdonos	Gp	Processzor	Mag	GFLOPS
Adatbzis 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 Dniel	1	E7300	2	21.2
Mnyoki dm	2	E5405		
Mnyoki dm	1	Q6600	4	38.4
Mnyoki dm	1	X4 920	4	52.9
Mreg Balzs	1	T4500	2	
sszesen	255	-		

Table 5: A szmtsaink sorn felhasznlt szmtgpek adatai.

Sorszm	Futsi id	$\gamma_{24,m}$
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

Table 6: A γ_{24} rvidtett naplja.

n	Futsi id (nap)	Rszfeladatok szma
25	26	435
26	70	435
27	316	435
28	1130	2 001
29	6733	15 119

Table 7: A szmtsok sorn mrt futsi idk sszege s az sszes rszfeladatok szma.

kiszmtsakor.

A dolgozat tdik rszben ismertettk eredmnyeinket (az jonnan megadott γ s ζ rtkeket, az j algoritmus s a rgi algoritmus futsi idejnek sszehasonltst), valamint nhny technikai rszletet a szmtsok elvgzsnek krlmnyeirl (naplfjlok s elrhetsgk, felhasznlt szmtgpek tpusa, rendelkezsre ll maximlis szmtsi teljestmny).

A dolgozatban lert eredmnyeket a [8, 9] cikkekben publikltuk.

Az eddig elrt eredmnyeket szeretnnk a ksbbiekben felhasznlni focisorozatok s szuper-versenyek sorozatainak ellenrzse sorn, mivel ezeknek a problmknak rsze a grfok foksorozatainak ellenrzse.

References

[1] Chodum, S. A.: *A simple proof of the Erds-Gallai theorem on graph sequences.* Bull. Austral. Math. Soc. **33,** (1986), 67–70.

[2] Erdős, P., Gallai, T.: *Grfok elrt fok pontokkal.* Mat. Lapok **11,** (1960) 264–274.

[3] Erds, P.; Mikls, I.; Toroczkai, Z.: *A simple Havel-Hakimi type algorithm to realize graphical degree sequences of directed graphs.* Elec. J. Comb. **17,** (2010) R66.

[4] 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.

[5] Havel, V.: A remark on the existence of finite graphs (cseh); *Časopis Pěst. Mat.* **80,** (1955), 477–480.

[6] Ivnyi, A.: *Reconstruction of complete interval tournaments.* Acta Univ. Sapientiae, Inform., **1,** (2009) 71–88.

[7] Ivnyi, A.: *Reconstruction of complete interval tournaments. II.* Acta Univ. Sapientiae, Math., **2,** (2010) 47–71.

[8] Ivnyi, A., Lucz, L., Str, P.: *On the Erds-Gallai and Havel-Hakimi algorithms.* Acta Univ. Sapientiae, Inform., **3,** (2011) elfogadva.

[9] Ivnyi, A., Lucz, L., Str, P., Pirzada, S.: *Parallel Erds-Gallai algorithm.* CEJOR, **kzirat**.

[10] Ivnyi, A., Pirzada, S.: *Comparison based ranking.* In (ed. A. Ivnyi): Alg. of Inform., **3,** AnTonCom, Budapest 2011, 1209–1256.

[11] Jacobson, V.: *Congestion avoidance and control.* In: Proc. of SIGCOMM 88, Stanford, CA, (1988), ACM, 314–329.

[12] Kleitman, D. J., Wang, D.L.: *Algorithms for constructing graphs and digraphs with given valences and factors.* Disc. Mat., **6,** (1973) 79–88.

[13] LaMar, M. D.: *Algorithms for realizing degree sequences of directed graphs.* arXiv (2009). Elrhet: http://arxiv.org/abs/0906.0343

[14] 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.

[15] Sloane N. J. A.: *The number of degree-vectors for simple graphs.* In (ed. N. J. A. Sloane): The On-line Encyclopedia of the Integer Sequences (2011).

[16] Tripathi, A., Vijay, S.: *A note on a theorem of Erdős & Gallai.* Disc. Math. **265,** (2003) 417–420.

[17] Tripathi, A., Venugopalan, S., West, D. B.: *A short constructive proof of the Erds-Gallai characterization of graphic lists.* Disc. Math. **310,** (2010) 833–834.