MŰSZAKI SZEMLE
43. szám, 2008.

Szerkesztőbizottság elnöke / President of Editing Committee
Dr. Köllő Gábor

Szerkesztőbizottság tagjai / Editing Committee
Dr. Balázs L. György – HU,
Dr. Biró Károly Ágoston – RO,
Dr. Csibi Vencel-József – RO,
Dr. Fedák László – UA,
Dr. Kása Zoltán – RO,
Dr. Kászonyi Gábor – HU,
Dr. Majdik Kornélia – RO,
Dr. Maros Dezső – RO,
Dr. Nagy László – RO,
Dr. Péics Hajnalka – YU,
Dr. Puskás Ferenc – RO,
Dr. Szalay György – SK,
Dr. Turchany Guy – CH

Kiadja / Editor
Erdélyi Magyar Műszaki Tudományos Társaság – EMT
Societatea Maghiară Tehnico-Ştiinţifică din Transilvania
Ungarische Technisch-Wissenschaftliche Gesellschaft in Siebenbürgen
Hungarian Technical Scientific Society of Transylvania

Felelős kiadó / Managing Editor
Dr. Köllő Gábor

A szerkesztőség címe / Address
Romania
400604 Cluj, Kolozsvár
B-dul 21. Decembrie 1989., nr. 116.
Tel/fax: 40-264-590825, 594042
Levélcím: RO – 400750 Cluj, C.P. 1-140.

Nyomda / Printing
Incitato Kft.

ISSN 1454-0746

CNCSIS által elismert folyóirat
Revistă acreditată de CNCSIS

www.emt.ro
emt@emt.ro

Tartalomjegyzék – Content– Cuprins

A kiadvány megjelenését támogatta:

COMMUNITAS ALAPÍTVÁNY
Communitas Alapítvány – Kolozsvár

eurotrans alapítvány
Eurotrans Alapítvány – Kolozsvár

De Bruijn-gráfok mint hálózati modellek

De Bruijn Graphs as Network Model

Grafuri De Bruijn ca modele de rețele

Dr. KÁSA Zoltán

Babeş-Bolyai Tudományegyetem, Kolozsvár
kasa@cs.ubbcluj.ro

ABSTRACT

A De Bruijn word for given q and k is a word over an alphabet with q letters, containing all k-length words exactly once. The length of such a word is q^k+k-1.

For a q-letter alphabet A the De Bruijn graph is defined as: B(q,k)= (V(q,k), E(q,k)) with $V(q,k)=A^k$ as the set of vertices, and $E(q,k)= A^{k+1}$ as the set of directed arcs. There is an arc from $x_1x_2...x_k$ to $y_1y_2... y_k$ if $x_2x_3...x_k=y_1y_2... y_{k-1}$, and this arc is denoted by $x_1x_2...x_k y_k$. In the De Bruijn graph B(q,k) a path (i. e. a walk with distinct vertices) $a_1a_2...a_k$, $a_2a_3...a_{k+1}$, ..., $a_{r-k+1}a_{r-k+2}...a_r$ (r>k) corresponds to an r-length word $a_1a_2...a_ka_{k+1}...a_r$, which is obtained by maximal overlapping of the neighboring vertices. For example in B(2,3) the path 001, 010, 101 corresponds to the word 00101. De Bruijn graphs are suitable models for networks.

Every maximal length path in the graph B(q,k) (which is a Hamiltonian one) corresponds to a De Bruijn word. In the directed graph B(q,k) there always exists an Eulerian circuit because it is connected and all its vertices have the same indegree and outdegree q. An Eulerian circuit in B(q,k) is a Hamiltonian path in B(q,k+1), which always can be continued in a Hamiltonian cycle.

The main aim of this paper is to study different properties of the De Bruijn graphs, their relations to the complexity of words, and to formulate a new variant of the conjecture given in [1,3] on the number of arc-independent Hamiltonian cycles in De Bruijn graphs. A morphism μ is defined on words over A = {0,1,..., q-1} such that μ (0) = 0; μ (i) = i+1, if $1 \le i < q-1$; and μ (q-1) = 1.

Conjecture: In the De Bruijn graph B(q,k) for q>2, k>1, there exists a Hamiltonian cycle H_0 such that the Hamiltonian cycles $H_1, H_2, ..., H_{q-2}$, obtained from H_0 by using the morphisms μ^k (k=1,2,..., q-2), together with H_0 are arc-disjoint Hamiltonian cycles.

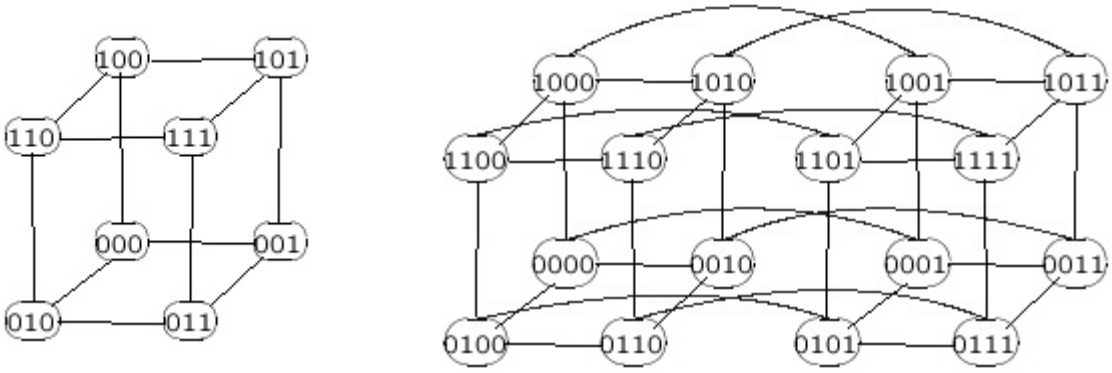
Kulcsszavak: De Brujin-gráfok, hálózati modell, élfüggetlen Hamilton-körök

HÁLÓZATI MODELLEK

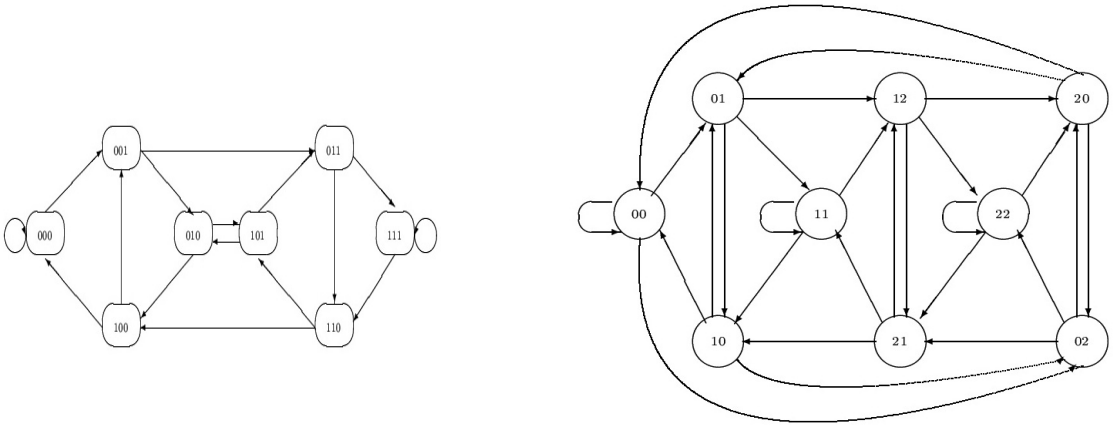
Adott *q*-betűs *A* ábécé fölött értelmezzük a *De Bruijn-gráfot*. A *B(q,k)= (V(q,k), E(q,k))* gráfot, ahol a gráf csúcsainak halmaza $V(q,k)=A^k$, azaz az *A* fölötti *k* hosszúságú szavak halmaza, irányított éleinek halmaza pedig $E(q,k)= A^{k+1}$, azaz a *k*+1 hosszúságú szavak halmaza, *De Bruijn-gráfnak* nevezzük. Akkor van él az $x_1x_2...x_k$ csúcsból az $y_1y_2... y_k$ csúcsba, ha $x_2x_3...x_k=y_1y_2... y_{k-1}$, és ezt az élet $x_1x_2...x_k y_k$-val jelöljük. A *B(q,k)* gráfban az $a_1a_2...a_k$, $a_2a_3...a_{k+1}$, ..., $a_{r-k+1}a_{r-k+2}...a_r$ ($r>k$) útnak (olyan séta, amelyben a csúcsok mind különbözőek) megfeleltethetünk egy *r*-hosszúságú $a_1a_2... a_ka_{k+1}...a_r$ szót, amelyet úgy kapunk meg, hogy a szomszédos csúcsokat maximálisan átfedjük. Például *B*(2,3)-ban a 001, 010, 101 útnak megfelel a 00101 szó.

A De Bruijn-gráfok a számítógép-hálózatok jó modelljei a hiperkockákhoz (1. ábra) hasonlóan. Ebben az esetben eltekintünk az élek irányításától, kiküszöböljük a párhuzamos éleket és a hurkokat.

Az *n*-dimenziós hiperkocka esetében a gráf csúcsai *n* hosszúságú bináris szavak, és két csúcs akkor van összekötve egy nem irányított éllel, ha a hozzájuk rendelt szavak csupán egyetlen bitben különböznek egymástól. A hiperkockák és De Bruijn-gráfok azért alkalmasak hálózati modellként, mert sok jó tulajdonsággal rendelkeznek. Ezek közül egyik legfontosabb, hogy többszörösen összefüggőek, azaz egyes élek jelölte kapcsolatok megszünte esetén is megőrzik összefüggőségüket. Cikkünkben egy olyan sejtésről lesz szó, amelyik kimondja, hogy a *B(q,k)* De Bruijn-gráfokban *q*-1 olyan élfüggetlen út van, amelyek a gráf minden pontján átmennek, tehát, ha ezek közül *q*-2 úton sérülnek élek, akkor is el lehet jutni minden csúcsból minden csúcsba.



1. ábra. Három- és négydimenziós hiperkockák



2. ábra. *B(*2,3*)* és *B(*3,2*)* De Bruijn-gráfok

DE BRUIJN-GRÁFOK TULAJDONSÁGAI

A *B(q,k)* gráf minden maximális hosszúságú irányított útja Hamilton-út, azaz tartalmazza a gráf összes csúcsát. Egy ilyen Hamilton-út mindig folytatható, hogy Hamilton-kör legyen, mivel utolsó és első csúcsa között van irányított él. Egy Hamilton-útnak megfelelő szót De Bruijn-szónak nevezünk. Ez tartalmazza az összes k hosszúságú szót az A ábécé fölött, és a legrövidebb ilyen tulajdonsággal rendelkező szó. A *b(q,k)*-val jelölt De Bruijn-szó hossza q^k+k-1. A De Bruijn-gráfokban mindig létezik zárt Euler-vonal is (olyan zárt séta, amely tartalmazza a gráf minden élét, egyszer és csakis egyszer). A következőkben felsoroljuk a De Bruijn-gráfok néhány fontos tulajdonságát.

1. A *B(q,k)* gráf tetszőleges zárt Euler-vonala a *B(q,k+*1*)* gráfban Hamilton kör, ha $k\geq 1$ és $q\geq 2$ [1].
2. Ha $q>2$, akkor a *B(q,k)* gráfból egy Hamilton-kör éleinek törlése után kapott gráf is összefüggő [7]. Itt összefüggőségen azt értjük, hogy az élek irányításának elhagyásával a gráf összefüggő, azaz bármely két pontja között van út.
3. Ha $q>2$, akkor a *B(q,k)* gráf bármely Hamilton-köre folytatható úgy, hogy zárt Euler-vonalat kapjunk [1].
4. Ha a $B(2,k)$ gráfban az $a_1a_2...a_k$, $a_2a_3...a_{k+1}$, …, $x_1x_2...x_k$ út Hamilton-út, akkor az $x_2x_3...x_k0$ és az $x_2x_3...x_k1$ a Hamilton-út csúcsai, és az egyik egyenlő az $a_1a_2...a_k$ csúccsal.
5. A *B(q,k)* gráf bármely két csúcsa között létezik k hosszúságú séta.
6. A *B(q,k)* gráf tartalmaz m hosszúságú irányított kört minden $1 \leq m \leq q^k$ esetben [8].

Sok algoritmus létezik De Bruijn-szó (így egyben Hamilton-út) generálására. Itt csupán Martin algoritmusát mutatjuk be [6]. Legyenek az ábécé betűi a 0,1,2,..., q-1 számok (a q alapú számrendszerben számjegyek), jelöljük ezeket sorra az $a_1,a_2,...,a_q$ jelekkel.

Az $a_1a_1...a_1$ (k-szor) szóból indulunk ki, és jobbról hozzáillesztünk egy-egy betűt a következők szerint: az olyan legnagyobb indexű a_k-val folytatjuk a szót, amelyre fennáll az, hogy az utolsó k betűből alkotott részszó (beleértve az a_k-t is) még nem szerepel a szóban. Addig folytatjuk, ameddig csak lehetséges. Be lehet bizonyítani [6], hogy csak akkor nem lehet már folytatni, amikor minden k hosszúságú szó már szerepel pontosan egyszer a szóban. Természetesen, ekkor a szó hossza q^k+k-1.

```
MARTIN(A):
for i:=1,2, …, k  do b_i := a_1 endfor
i := k
repeat
  megáll := true
  j := n
  while j>1 do
        if b_{i-k+2}b_{i-k+3}…b_ia_j  nem részszava b_1b_2…b_i-nek
           then  i := i+1
           b_i:=a_k
                 megáll := false
                 exit while
            else  j := j-1
  endwhile
until megáll
return b
```

Példa. Legyen A ={0,1}. Keresünk egy *b(2,3)* De Bruijn-szót.
000-val kezdünk.
Hozzáilleszthetjük az 1-est. A kapott szó: 0001.
Hozzáilleszthetjük az 1-est. A kapott szó: 00011.
Hozzáilleszthetjük az 1-est. A kapott szó: 000111.
Most már csak a 0-t illeszthetjük, mivel 111 már szerepel (átfedéssel). A kapott szó: 0001110.
Hozzáilleszthetjük az 1-est. A kapott szó: 00011101.
Ismét csak a 0-t illeszthetjük, mert 011 már szerepel a szóban. A kapott szó: 000111010.
Ismét csak a 0-t illeszthetjük, mert 101 már szerepel (átfedéssel). A kapott szó: 0001110100.
Nem lehet folytatni sem 1-gyel, sem 0-val. Tehát, a keresett szó 0001110100.

ÉLFÜGGETLEN HAMILTON-KÖRÖK DE BRUIJN-GRÁFOKBAN

Két Hamilton-kört élfüggetlennek nevezünk, ha nincs közös élük. Egy olyan sejtéssel foglalkozunk (1. sejtés), amelyet több évtizede nem sikerült bizonyítani. Átfogalmazásunk (2. sejtés) segíthet a bizonyítás megtalálásában.

1. sejtés. [1,3] A *B(q,k)* gráfban, ahol $q\geq 2$ és $k>1$, létezik q-1 élfüggetlen Hamilton-kör.

Értelmezzük a μ morfizmust az A = {0,1,…, q-1} ábécé betűiből képzett szavakon:

$\mu(0) = 0$
$\mu(i) = i+1$, ha $1\leq i <q-1$
$\mu(q-1) = 1$.

Könnyű belátni, hogy $\mu^{q-1}(u) = u$ tetszőleges u szóra.

Példa. Legyen adott a w = 00102113230331220 szó az A = {0,1,2,3} ábécé felett. Ekkor

$\mu(w)$ = 00203221310112330
$\mu^2(w)$ = 00301332120223110
$\mu^3(w)$ = 00102113230331220.

Egy De Bruijn-szóból Hamilton-kör kapható (és fordítva). Legyen H_0 egy ilyen Hamilton-kör. Alkalmazva a μ^k morfizmust minden k=1,2,… q-2 értékre a H_1, H_2, …, H_{q-2} Hamilton-köröknek megfelelő De Bruijn-szavakat kapunk.

2. sejtés. [4] A *B(q,k)* gráfban $q>2$, $k>1$ esetén létezik egy H_0 Hamilton-kör úgy, hogy a belőle a μ^k (k=1,2,…q-2) morfizmusok által kapott H_1, H_2, ..., H_{q-2} Hamilton-körök a H_0-val együtt élfüggetlenek.

Példák élfüggetlen Hamilton-körökre (De Bruijn-szavak segítségével):

B(3,2): H_0: 0011220210
H_1: 0022110120

B(3,3) H_0: 000100210110222020121112212 00
H_1: 000200120220111010212221121 00

B(4,2) H_0: 00102113230331220
H_1: 00203221310112330
H_2: 00301332120223110

B(5,2) H_0: 0010211204142243033231344 0
H_1: 0020322301213314044342411 0
H_2: 0030433402324421011413122 0
H_3: 0040144103431132022124233 0.

A 2. sejtésbeli H_0 Hamilton-körnek megfelelő De Bruijn-szót *reprezentatív De Bruijn-szónak* nevezzük. A sejtés tehát az, hogy mindig létezik reprezentatív De Bruijn-szó. Be lehet bizonyítani, hogy a Martin-algoritmus soha nem generál reprezentatív De Bruijn-szót [5].

Egyéb reprezentatív De Bruijn-szavak:

b(6,2)=0010211204131403325235505154534422430
b(7,2)=0010211204130614031505516225235343644246332654566 0
b(4,3)=0001002101102012023103013111211302212320313230033321331223303222 00

Számítógéppel sok esetben ki lehet próbálni sejtésünket. A bizonyításhoz jó lenne olyan algoritmust találni, amelyik bizonyíthatóan reprezentatív De Bruijn-szavakat generál.

KÖNYVÉSZET

[1.] Bond, J., Iványi A., Modelling of interconnection networks using De Bruijn graphs, *Third Conference of Program Designer*, Ed. A. Iványi, Budapest, 1987, 75–87.

[2.] De Bruijn, N. G., A combinatorial problem, *Nederl. Akad. Wetensch. Proc.* 49 (1946), 758–764.

[3.] Gire, S., Réseaux d'interconnexion, Option Maitrise Informatique, 1996–97, http://fastnet.univ-brest.fr/~gire/COURS/OPTION_RESEAUX/node48.html

[4.] Kása Z., A conjecture on arc-disjoint Hamiltonian cycles in De Bruijn graphs, *5th MaCS (5th Joint Conference on Mathematics and Computer Science)*, Debrecen, 9–14 June 2004.

[5.] Kása Z., A conjecture on arc-disjoint Hamiltonian cycles in De Bruijn graph, *Grant AR* 13/2006, pp. 8.

[6.] Martin, M. H., A problem in arrangements, *Bull. A.M.S.* 40 (1934), 859–864.

[7.] Vörös N., On the complexity measures of symbol-sequences, *Conference of Young Programmers and Mathematicians,* Ed. A. Iványi, Budapest, Eötvös Loránd University, 1984, 43–50.

[8.] Yoeli, M., Binary ring sequences, *Amer. Math. Monthly* (1962) 852–855.

Algoritmusok hatékonyságának növelése a bináris keresés elvének alkalmazásával

Improving the Performance of Algorithms Using the Principles of Binary Search

Mărirea eficienței algoritmilor prin folosirea principiului căutării binare

Dr. IONESCU Klára, Drd. PĂTCAŞ Csaba

Babeş-Bolyai Tudományegyetem, Kolozsvár
clara@cs.ubbcluj.ro, patcas.csaba@gmail.com

ABSTRACT

When analyzing the performances of our algorithms, we are interested in two important things: the time of execution and the dimension of the needed memory space. Obviously, an algorithm will be more efficient than another, if the first one has a shorter time of execution and it needs a smaller memory space. Thus, when we want to increase the performance of an algorithm, we focus either on the time of its execution or on the needed memory space. If we are lucky, maybe we succeed in improving both of these properties of our algorithm.

In this article we present a few algorithmic/programming task and we propose algorithms designed with the Divide and Conquer method, more exactly with the idea of the binary search, in order to obtain a better execution time.

Kulcszavak: Bináris keresés, hatékonyság, feladatok

1. A BINÁRIS KERESÉS

Bizonyos feladattípusok esetében a megoldásként javasolt *lineáris* algoritmus egy *logaritmikus* bonyolultságúval helyettesíthető, ha felhasználjuk a *bináris keresés* elvét. Mielőtt rátérnénk adott feladatok megoldását képező algoritmusok bemutatására, lássuk az *Oszd meg és Uralkodj* elvére alapuló bináris keresés klasszikus algoritmusát!

1.1. Feladat – Keresés

Adva van egy `n` *egész számból álló, növekvően rendezett sorozat. Állapítsuk meg egy adott szám helyét a sorozatban! Ha a keresett szám nem található meg a sorozatban, a helynek megfelelő változó értéke legyen* 0*!*

Megoldás

Legyen a rendezett sorozat $x_1 < x_2 < ... < x_n$. Egy bizonyos értéket keresünk (*keresett*), amelynek a helye ismeretlen. Az *Oszd meg és Uralkodj* módszer alapötletére támaszkodva a sorozatot két részre osztjuk: az x_1, ..., $x_{közép-1}$ és az $x_{közép+1}$, ..., x_n részsorozatokra. Vegyük észre, hogy az $x_{közép}$ elem nem tartozik egyik részsorozathoz sem. Ezt az elemet külön vizsgáljuk és a vizsgálat eredményétől függően az algoritmus befejeződik vagy az eredeti elképzeléshez hasonlóan folytatódik valamelyik részsorozatra. A következő esetek fordulhatnak elő:

1. $keresett = x_{közép} \Rightarrow$ *keresett* a sorozatban a *közép* helyen található;
2. $keresett < x_{közép} \Rightarrow$ mivel a sorozat rendezett, a `keresett` számot a sorozat első (x_1, ..., $x_{közép-1}$) felében keressük tovább;
3. $keresett > x_{közép} \Rightarrow$ a `keresett` számot a sorozat második ($x_{közép+1}$, ..., x_n) felében keressük tovább.

Következésképpen, a `keresett` elem megkeresése átalakul *egyetlen* feladattá: keressük az elemet *vagy* az x_{bal}, ..., $x_{közép-1}$ sorozatban, vagy az $x_{közép+1}$, ..., x_{jobb} sorozatban. Előbb bemutatjuk az algoritmus rekurzív változatát:

```
Algoritmus Bin_keres_Rekurzívan(bal,jobb,hely):
{ Bemeneti paraméterek: bal, jobb – a vizsgált részsorozat első és utolsó indexe }
{ Kimeneti paraméterek: hely – a keresett elem indexe az eredeti sorozatban }
{ az algoritmust az 1, n bemeneti paraméterértékekre hívjuk meg }
  Ha bal > jobb akkor
    hely ← 0                    { ha keresett nem található meg a sorozatban, a hely változó értéke 0 }
  különben
    közép ← ⌊(bal + jobb)/2⌋    { felosztás: kiszámítjuk a sorozat közepén található elem indexét }
    Ha keresett = x[közép] akkor
      hely ← közép              { keresett a sorozatban a közép helyen található }
    különben
      Ha keresett < x[közép] akkor
        Bin_keres_Rekurzívan(bal,közép-1,hely)
        { az x_bal, ..., x_közép-1 részsorozatban keresünk tovább }
      különben
        Bin_keres_Rekurzívan(közép+1,jobb,hely)
        { az x_közép+1, ..., x_jobb részsorozatban keresünk tovább }
      vége(ha)
    vége(ha)
  vége(ha)
Vége(algoritmus)
```

E feladat esetében is létezik egy iteratív megoldás, amely a végrehajtás idejét és a szükséges memóriát tekintve is hatékonyabb, mivel nincs szükség veremre és veremmel kapcsolatos műveletekre.

```
Algoritmus Bin_Keres_Iteratívan(n,x,keresett,hely):
  bal ← 1
  jobb ← n
  hely ← 0
  Amíg (hely = 0) és (bal ≤ jobb) végezd el:
    közép ← ⌊(bal + jobb)/2⌋
    Ha x[közép] = keresett akkor
   hely ← közép
    különben
      Ha x[közép] > keresett akkor
        jobb ← közép – 1
      különben
        bal ← közép + 1
      vége(ha)
    vége(ha)
  vége(amíg)
Vége(algoritmus)
```

2. ALGORITMUSOK BONYOLULTSÁGÁNAK CSÖKKENTÉSE A BINÁRIS KERESÉS ELVÉNEK ALKALMAZÁSÁVAL

A következőkben bemutatunk néhány feladatot, amelyeknek megoldásaiban felhasználjuk a bináris keresést. Nem olyan feladatokra gondolunk, amelyekben a keresés megjelenik mint explicit részfeladat, hanem olyanokra, amelyekben az eredmény egy olyan érték, amelynek értéktartományát ismerjük, ugyanakkor lehetséges a „találgatás” a bináris keresés algoritmusának alkalmazásával.

2.1. Feladat – Összeg

Legyen egy n elemű ($3 \le n \le 100000$) *különböző természetes számokat tartalmazó sorozat és az S természetes szám. Válasszunk ki az adott sorozatból három elemet, amelyeknek az összege pontosan S!*

Megoldás

Észrevesszük, hogy részletösszegeket kell számítanunk, de pontosan három elem összegét hasonlítjuk *S*-sel.

A feladat megoldható három egymásba ágyazott **Minden** ciklussal is. Sajnos, az *n* értéke miatt ez a megoldás nem biztos, hogy befér egy esetleges időhatárba.

Ha a megoldásba beépítjük a bináris keresést, a bonyolultság $O(n^2 \log n)$ lesz:

- Rendezzük az adott sorozatot.
- Két **Amíg** ciklussal kiválasztunk a sorozatból két elemet (legyen ezeknek indexe *i* és *j*).
- Megkeressük az $S – a_i – a_j$ értéket a bináris keresést alkalmazva.
- A megoldást tovább javítjuk (pl., ha a_i értéke meghaladja *S*-t, kilépünk az első **Amíg**-ból stb.).

```
        Algoritmus Generál(n,a,S,i,j,k):
{ Bemeneti paraméterek: n – a sorozat mérete, a – az adott sorozat, S - az adott összeg }
{ Kimeneti paraméterek: i,j,k – három index a sorozatban (a megfelelő indexű elemek összege S) }
          megvan ← hamis
          i ← 1
          Amíg not megvan és (i ≤ n) és (a_i < S) végezd el:
            j ← i + 1
            Amíg not megvan és (j ≤ n) és (a_i + a_j < S) végezd el:
              BinKeres(a,j+1,n,S-a_i-a_j,k) { ha S-a_i-a_j megtalálható a sorozatban, a j+1 indexű elem után, }
              Ha k ≠ 0 akkor                { akkor k értéke egy index, különben k értéke 0 }
                megvan ← igaz
              vége(ha)
              j ← j + 1
            vége(amíg)
            i ← i + 1
          vége(amíg)
Vége(algoritmus)
```

2.2. Feladat – Labirintus

Egy labirintust egy $n \times n$ méretű négyzetes tömbbel kódolunk. Az (*i, j*) *helyen levő* 1 *érték falat jelent, a* 0 *szabad helyet. Adottak még a kiindulási és az érkezési pontok koordinátái.*

Írjuk ki annak a legnagyobb területű négyzetnek a méretét, amely eltolható a kiindulási ponttól az érkezési pontig, tudva, hogy egy k oldalhosszúságú négyzet minden pillanatban k^2 helyet foglal el az eredeti tömbben és csak vízszintesen, vagy függőlegesen mozgatható úgy, hogy ne haladjon át falon.

Megoldás

Ha a négyzet oldalhossza 1 volna, a feladatot megoldhatnánk a *dinamikus programozás* módszerével, vagy egy egyszerű, szélességi bejáráshoz hasonló algoritmussal.

A nehézséget az okozza, hogy minden lépésben ki kell kerülnünk a falat. Jó lenne, ha egy *k* oldalhosszúságú négyzet eltolása előtt nem kellene megvizsgálnunk *k* szomszédos helyet, (mondjuk a jobb oldallal szomszédosakat, ha jobbra akarunk mozdulni). Kiszámíthatnánk előre, minden pozícióra a legnagyobb oldalhosszúságot, aminek megfelelő négyzetet az illető helyről el lehet tolni mind a négy irányban. Tehát, megpróbálunk eltolni egy 1, 2, 3 stb. oldalhosszú négyzetet, amíg a költöztetés lehetséges, majd kiválasztjuk a legnagyobb méretű négyzetet amit el lehetett tolni.

Ha figyelmesebben elemezzük a fenti gondokat, megállapíthatjuk, hogy ezek kiküszöbölhetők. Olyan algoritmust javasolunk, amely kevesebb memória használatával, ugyanakkor gyorsabban dolgozik. Észrevesszük, hogy egy adott pozícióból valamely irányba tolható négyzet mérete megkapható a kiindulási és a végső pozícióba elhelyezhető négyzetek méreteinek minimumából. Ezek a négyzet-méretek megkaphatók $O(n^2)$ időben, a dinamikus programozás módszerének segítségével.

Az algoritmus javítása abban áll, hogy felhasználjuk a *bináris keresés*t az *oldalhossz megkeresésére*. Észrevesszük, hogy a legnagyobb (elvben eltolható) négyzetnek a mérete *n*, a legkisebbé 1. Tehát az 1 és *n* közötti számok között meg kell találnunk azt a legnagyobbat, amelynek egy eltolható négyzet felel meg. Legelőbb egy *n*/2 oldalhosszú négyzetet próbálunk eltolni; ha ez sikerült, megpróbálunk egy 3 *n*/4 oldalhosszút, egyébként egy *n*/4 oldalhosszút stb. Minden kipróbálandó oldalhossz esetében (összesen log *n* ilyen hosszúságunk lesz) megoldjuk a feladatot egy szélességi bejárással.

Ezáltal a feladatot megoldó algoritmus bonyolultsága $O(n^2 \cdot \log n)$.

Szükségünk lesz két darab egydimenziós segédtömbre, amelyeket konstanstömbökként fogunk használni: dx=(-1,0,0,1) és dy=(0,-1,1,0). A tömb elemeit a négyzetek megkeresésekor a következő pozíció megadására használjuk. Egy adott helyről 4 irányba lehet menni, ennek megfelelően a két segédtömb indexeinek jelentése: 1 – fel, 2 – balra, 3 – jobbra, 4 – le.

Példa

```
10 2 1 6 5
1 1 1 1 1 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 1 0 0 0 0 0
1 1 0 0 1 1 1 1 1 1
1 1 0 0 0 0 0 1 1 1
1 1 0 0 0 0 0 1 1 1
1 1 1 1 0 1 1 1 1 1
1 1 1 1 0 1 1 1 1 1
1 1 1 1 0 0 0 0 1 1
```

Eredmény

```
2
```

```
Algoritmus Épít_MaxNégyzet(n,a,MaxNégyzet):
{ Bemeneti paraméterek: n a labirintus mérete, a a labirintusnak megfelelő kétdimenziós tömb }
{ Kimeneti paraméter: MaxNégyzet segédtömb, minden pozícióra tárolja a legnagyobb négyzet oldalhosszát, }
{ aminek bal felső sarka az adott pozícióba kerülhet MaxNégyzet méretei:[0..n+1,0..n+1], }
{ ahol a 0 és n+1 indexű sorok és oszlopok bekeretezik a labirintust ahhoz, hogy ne léphessünk ki belőle }
{ feltöltjük a MaxNégyzet tömböt 0-val }
  Minden q=n,1,-1 végezd el:
    Minden w=n,1,-1 végezd el:
      Ha a_qw = 1 akkor
        MaxNégyzet_qw ← 0              { fal }
      különben
        MaxNégyzet_qw ← 1 + min(MaxNégyzet_q+1,w, MaxNégyzet_q,w+1, MaxNégyzet_q+1,w+1)
      vége(ha)
    vége(minden)
  vége(minden)
Vége(algoritmus)

Algoritmus BF(MaxNégyzet,méret):
{ Függvény típusú algoritmus: szélességi bejárás (Breadth First) }
{ Bemeneti paraméter: MaxNégyzet, méret – az eltolásra javasolt segédtömb mérete }
{ Kimenet: igaz, ha el tudtunk jutni a végső pozícióba ezzel a mérettel }
  Ha MaxNégyzet_x1,y1 ≥ méret akkor
  { feltöltjük a volt segédtömböt a hamis értékkel }
  { volt – kétdimenziós segédtömb, amelyben nyilvántartjuk, hogy mely mezőket érintettük }
    volt_x1,y1 ← igaz
    sor.első ← 1          { sor segédtömb, a szélességi bejáráshoz szükséges várakozási sor }
    sor.utolsó ← 1
    sor.v_1.x ← x1
    sor.v_1.y ← y1
    Amíg (sor.első ≤ sor.utolsó) és (nem volt_x2,y2) végezd el:
      AktX ← sor.v_első.x
      AktY ← sor.v_első.y
      Minden q=1,4 végezd el:
        Ha nem volt_AktX+dx_q,AktY+dy_q és (MaxNégyzet_AktX+dx_q,AktY+dy_q ≤ méret) akkor
          sor.utolsó ← sor.utolsó + 1
          sor.v_utolsó.x ← AktX + dx_q
          sor.v_utolsó.y ← AktY + dy_q
          volt_v_utolsó.x,v_utolsó.y ← igaz
        vége(ha)
      vége(minden)
      sor.első ← sor.első + 1
    vége(amíg)
    BF ← volt_x2,y2
  különben
    BF ← hamis
  vége(ha)
Vége(algoritmus)

Algoritmus Bináris_Keresés(n,MaxNégyzet,lent):
{ Bemeneti paraméter: a keresés után megoldjuk a feladatot a MaxNégyzet felhasználásával }
{ Kimeneti paraméter: lent – a megtalált eltolható négyzet oldalhossza }
  lent ← 1
  fent ← n
  Amíg lent < fent végezd el:
    közép ← [(lent+fent)/2]
    Ha BF(MaxNégyzet,közép) akkor
      lent ← közép
    különben
      fent ← közép - 1
    vége(ha)
  vége(amíg)
  Ha nem BF(MaxNégyzet,lent) akkor
    lent ← 0                  { nincs megoldás }
  vége(ha)
Vége(algoritmus)
```

2.3. Feladat – Út

Adott egy irányítatlan gráf. Minden éléhez hozzárendelünk egy hosszúságot és egy veszélyességi faktort. Írjunk ki egy olyan utat, amely az 1-es csomópontot összeköti az n-edikkel és amely úthoz tartozó élek legnagyobb veszélyességi faktora a lehető legkisebb! Ha több ilyen út létezik, akkor a legrövidebbet kell megtalálnunk.

Példa

csomópontok száma = 4, élek száma = 5

él	hossz	veszélyességi faktor
[1, 2]	1	5
[1, 3]	2	5
[1, 4]	1	10
[2, 4]	1	5
[3, 4]	2	5

Eredmény:

Út hossza: 2

Út: 1–2–4

Megoldás

Van megoldás a dinamikus programozás módszerével. „Összerakjuk” a veszélyességi faktort a költséggel: olyan valós számokkal dolgozunk, amelyeknek egész része a veszélyességi faktor, törtrésze pedig a hosszúság. Ennek az algoritmusnak bonyolultsága $O(n\ maxVeszélyességiFaktor)$ lenne.

Ennek a feladatnak a megoldásában a *bináris keresés*t a veszélyességi faktor függvényében végezzük. Az éleket a veszélyességi faktor szerint rendezzük és a keresés közben minden kiválasztott értékre meghatározzuk a legrövidebb utat *Dijkstra* algoritmusával úgy, hogy csak a bináris kereséssel kiválasztott értéknél kisebb vagy egyenlő veszélyességi faktorral rendelkező éleket vesszük figyelembe.

Ennek a megoldásnak bonyolultsága $O(n^2 \log maxVeszélyességiFaktor)$, ami javítható *Dijkstra* algoritmusának hatékonyabb implementálásával.

Az alábbi algoritmus végrehajtása előtt feltöltjük a `hossz` és a `veszély` tömböket `0`-val, majd beolvassuk az éleket és a megfelelő hosszúságokat és veszélyességi faktorokat. Meghatározzuk a veszélyességi faktorok maximumát (`mVeszély`). A `végtelen` egy megfelelően nagy érték, amelyet *Dijkstra* algoritmusában használunk.

```
Algoritmus Dijkstra(n,m,hossz,veszély,mVeszély):
{ Függvény típusú algoritmus }
{ meghatározza, hogy megoldható-e a feladat a javasolt maximális veszélyességi faktorral }
{ Kimenet: igaz, ha el tudtunk jutni a végső pozícióba a mVeszély veszélyességi faktorral }
{ Bemeneti paraméterek: n – a gráf csomópontjainak száma, m – a gráf csomópontjainak száma }
{ hossz – az élek hosszának sorozata, veszély – a veszélyességi faktorok sorozata }
{ mVeszély – a maximális veszélyességi faktor, a beolvasással párhuzamosan határozzuk meg }
{ ez lesz a bináris keresés felső határa az első iterációban }
{ feltöltjük a volt segédtömböt a hamis értékkel }
  Minden q=1,n végezd el:
    Ha (hossz_1q ≠ 0) { ha létezik él } és (veszély_1q ≤ mVeszély) { csak ezeket vesszük figyelembe }
    akkor
      d_q ← hossz_1q
      előző_q ← 1                          { az előző tömb megőrzi az úton található csomópontokat }
    különben
      d_q ← végtelen
    vége(ha)
  vége(minden)
  volt_1 ← igaz
  Minden q=2,n végezd el:
    iMin ← 0
    Minden w=1,n végezd el:
      Ha nem volt_w és ((iMin = 0) vagy (d_iMin > d_w)) akkor
        iMin ← w
      vége(ha)
    vége(minden)
    Ha d_iMin = végtelen akkor             { lehet, hogy nem juthatunk el az összes csomópontba }
      Dijkstra ← d_n ≠ végtelen                    { eljutottunk az n-edik csomópontba? }
      { ekkor kilépünk az algoritmusból, a függvény hamis értéket térít }
    vége(ha)
```

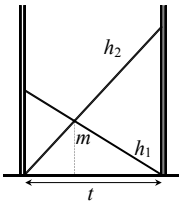
```
    voltiMin ← igaz
    Minden w=1,n végezd el:
      Ha nem voltw és (hossziMin,w ≠ 0) és (veszélyiMin,w ≤ mVeszély) és
          (dw > diMin + hossziMin,w) akkor
        dw ← diMin + hossziMin,w
        előzőw ← iMin
      vége(ha)
    vége(minden)
  vége(minden)
  Dijkstra ← igaz
Vége(algoritmus)

Algoritmus Út_kiír(csúcs):
  Ha csúcs ≠ 1 akkor
    Út_kiír(előzőcsúcs)
  vége(ha)
  Ki: csúcs,' '
Vége(algoritmus)

Algoritmus BinKeres:
  lent ← 1
  fent ← mVeszély
  Amíg lent < fent végezd el:
    m ← [(lent + fent)/2]
    Ha Dijkstra(m) akkor
      fent ← m
    különben
      lent ← m+1
    vége(ha)
  vége(amíg)
  Ha Dijkstra(lent) akkor
    Ki: 'Ut hossza:',dn
    Út_kiír(n)
  különben
    Ki: 'Nincs megoldás'
  vége(ha)
Vége(algoritmus)
```

2.4. Feladat – Deszkák

Két függőleges fal egymástól t távolságra található. Egy h_1 hosszúságú deszkát az egyik fal alapjától a másik falnak támasztunk. Egy h_2 hosszúságú deszkát a másik fal alapjától az első falnak támasztunk. A két deszka m magasságban érinti egymást egy pontban, amely valahol a két fal között található. Számítsuk ki t-t h_1, h_2 és m ismeretében (megengedett hibalehetőség 10^{-5}).



Megoldás

Észrevesszük, hogy a magasság, ahol a két deszka találkozik nő, ha a keresett *t* távolság csökken, és csökken, ha *t* nő.

Átfogalmazzuk a követelményt: keressük meg azt a legnagyobb *t* értéket, amelyre a magasság, ahol a két deszka találkozik ne legyen kisebb mint *m*!

Felhasználjuk a bináris keresést a *t* értékének megkeresésére. A kiindulási érték: $Min(h_1, h_2)/2$.

Ha ismerjük a *t*, h_1, h_2 értékeket, az érintkezési pont magasságát kiszámítjuk síkmértan ismeretekkel.

Ha a kiszámított magasság nagyobb mint az adott *m*, akkor növeljük *t*-t, különben csökkentjük.

A `Számol(h1,h2,t,sz)` algoritmus kiszámítja *sz*-ben a magasság értékét *t* aktuális közelítő értékére.

```
Algoritmus Számol(h1,h2,t,sz):
  x ← négyzetgyök(h1*h1-t*t)
  y ← négyzetgyök(h2*h2-t*t)
  sz ← (x*y)/(x+y)
Vége(algoritmus)

Algoritmus Deszkák(m,h1,h2,t):
  megvan ← hamis
  Amíg nem megvan végezd el:
```

```
      t ← (min + max)/2
      Számol(h1,h2,t,sz)
      Ha | sz - m | ≤ 0.0001 akkor
        megvan ← igaz
      különben
        Ha sz > m akkor
          min ← t
        különben
          max ← t
        vége(ha)
      vége(ha)
    vége(amíg)
Vége(algoritmus)
```

2.5. Feladat – Ládák

Költözik a múzeum. A tárgyakat kocka alakú, különböző méretű ládákba csomagolták. Kicsomagoláskor több személy dolgozik egyidőben, és a rendetlenség elkerülése végett, azokba a helyiségekbe, ahol kicsomagolás folyik, felszereltek egy futószalagot, amelyre az üres ládákat helyezik, a nyitott fedelükkel fölfele. A futószalag végéhez egy robotot állítottak, amelynek az a feladata, hogy összeszedje a ládákat és úgy helyezze egyiket a másikba (ha lehet) hogy végül a ládacsomagok száma a lehető legkisebb legyen. A robotot egy program irányítja úgy, hogy:

- *A ládákat az érkezésük sorrendjében szedi le a futószalagról.*
- *Az aktuális ládát csak egy nála nagyobb méretű ládába helyezi.*
- *Ha nincs olyan megkezdett csomag, amelybe elhelyezhető az aktuális láda, akkor ez a láda egy új csomag első ládája lesz.*
- *Egy megkezdett csomagba csak egyetlen ládát helyez, vagyis nem helyez két ládát egymás mellé, még akkor sem, ha ez egyébként lehetséges volna.*
- *Egy elhelyezett ládát, többé nem mozgat.*
- *Egy megkezdett csomagot nem helyez egy másik csomagba még akkor sem, ha ez egyébként lehetséges volna.*
- *Egyetlen ládát sem hagy figyelmen kívül.*

Írjunk programot, amely a ládák számának ($0 \le n \le 15\,000$*) és méreteinek (*$1 \le láda\_méret \le 10\,000$*) ismeretében meghatározza a csomagok lehetséges legkisebb számát, valamint minden csomag esetében az illető csomagban található ládákat.*

Példa

$n = 10$, *méretek* = (4, 1, 5, 10,7, 9, 2, 8, 3, 2)

Eredmény:

Ládacsomagok száma: 4,
Csomagok:
1. csomag = (4, 1)
2. csomag = (5, 2)
3. csomag = (10, 7, 3, 2)
4. csomag = (9, 8)

Megoldás

Ha félretesszük a mesét, észrevesszük, hogy a feladat tulajdonképpen azt kéri, hogy az adott sorozatot bontsuk fel minimális számú növekvő részsorozatra. A feladat megoldható egy *mohó algoritmus*sal, amely mindig a legkisebb olyan ládába csomagol, amelybe lehetséges. Észrevesszük, hogy így a ládacsomagokba utoljára elhelyezett ládák mérete növekvő sorozatot alkot, tehát a megfelelő csomag megkeresése lehetséges bináris kereséssel. Ugyanakkor az is világos, hogy nem egy ismert értéket kell megkeresnünk, hanem egy olyat, amely legkisebb az adott számnál nagyobbak között.

A gondot az adatok tárolása okozza, hiszen, ha a ládák csökkenő (vagy növekvő) sorrendben érkeznek, a következő két legrosszabb esettel állunk szemben:

1. Ha a ládák csökkenő sorrendben érkeznek, egyetlen csomagba befér minden láda, tehát egyetlen növekvő részsorozatunk lesz, aminek a hossza legtöbb 15 000.
2. Ha a ládák növekvő sorrendben érkeznek, akkor minden érkező láda új csomagnak felel meg, tehát legtöbb 15000 darab egy elemű részsorozatunk lesz.

A fentieket figyelembe véve egy 15 000×15 000 méretű tömböt kellene létrehoznunk, ami (ha lehetséges a választott programozási környezetben) nagyon nagy tárpazarlást jelent, hiszen még tárolnunk kell a csoma-

gok hosszát is egy legtöbb 15 000 elemű tömbben. A megoldást a dinamikus tárkezelés hozza: minden csomag egy verem típusú lista lesz, amelynek a feje az utoljára elhelyezett láda méretét tartalmazza. A bináris keresést a veremfejek sorozatán végezzük: ha nem találunk olyan ládát, amelybe az aktuális láda elhelyezhető, akkor új csomagot indítunk, különben elhelyezzük az aktuális ládát a megfelelő csomag tetejére. Végül kiírjuk a létrehozott verem típusú listák számát és ezeknek tartalmát. Megjegyezzük, hogy a csomagok tárolását statikusan is elvégezhetjük, egy 15 000 elemből álló vektor segítségével, melyben minden ládára tároljuk, hogy melyik ládát helyeztük el benne.

A keresést az alábbi algoritmussal végezzük, amelyben felismerhető a bináris keresés:

```
Algoritmus Keres(bal,jobb,új):
{ keressük a bal..jobb sorszámú csomagok közt azt, amelybe elhelyezhető az új méretű láda }
{ Bemeneti paraméterek: bal, jobb a ládacsomagok tömbszakaszának kezdő és végpontja }
{ az új méretű ládát fogjuk elhelyezni }
  Ha bal > jobb akkor                                          { sikertelen keresés, }
    jobb  ←  jobb + 1
    csomagokszáma ← jobb
    Helyez(csomagok,csomagokszáma,új)
    { új csomagot kezdünk, a jobb sorszámú után }
  különben
    Ha csomagok[bal]^.méret > új akkor                         { sikeres keresés }
      Helyez(csomagok,bal,új)          { az új méretű ládát a bal sorszámú csomagba tesszük }
    különben
      közép ← [(bal+jobb)/2]                                   { tovább keresünk }
      Ha új < csomagok[közép]^.méret akkor
        Keres(bal,közép,új)
      különben
        Keres(közép+1,jobb,új)
      vége(ha)
    vége(ha)
  vége(ha)
Vége(algoritmus)
```

3. KÖVETKEZTETÉSEK

A 3. és 4. feladatok megoldásai a következő általános szabályhoz vezetnek: ha egy maximális értéket kell meghatároznunk, amelynek olyan követelményeknek kell eleget tennie, amelyek ha teljesülnek egy bizonyos értékre, akkor biztosan teljesülnek az ennél kisebbekre, akkor a bináris kereséssel és a feltételek utólagos ellenőrzésével log *n* lépésben eredményt kapunk.

A szabály alkalmazható minimum esetében is.

Az elv alkalmazása az algoritmus végrehajtási idejének csökkentését eredményezi, de ehhez előbb be kell látnunk, hogy ez lehetséges, majd meg kell találnunk az alkalmazás módját.

KÖNYVÉSZET

[1.] Cormen T., Leiserson C., Rivest R.: *Algoritmusok*, Műszaki Könyvkiadó, Budapest, 1997.

[2.] Ionescu K.: *Bevezetés az algoritmikába,* Presa Universitară Clujeană, 2005.

[3.] Stroe M.: *Metode de reducere a complexităţii*, GInfo, Agora Media, Tg. Mureş, 13/7 (2003), 28–29 (román nyelven).

Fordító automaták

Automata for Compilers

Automate pentru compilatoare

Dr. KOVÁCS Lehel István

Sapientia – Erdélyi Magyar Tudományegyetem, Marosvásárhely
klehel@ms.sapientia.ro

ABSTRACT

The spread of compilers allows the high-level communication with computers. At the beginning programming could be accessed only by scientists but the development of applications, the programming made possible the access of other persons from the collectivity. This fact become possible due to the development of programming languages and compilers, which are able to translate a program written in a complicated language to cod which can be understand by the computer.

Our aim is to present a compiler development method, which uses special automata, automata-systems to realize a compiler.

Kulcsszavak: Fordítóprogram, automata, adaptív fordítóprogram, automata rendszer

1. KLASSZIKUS AUTOMATÁK

A fordítóprogramok egyik megvalósítási módszere automaták segítségével történik. [1.] így definiálta a *véges determinisztikus automatá*t:

1. definíció: Egy véges determinisztikus automata az $A = (Q, \Sigma, \delta, q_0, F)$ rendezett ötös, ahol:

- Q véges, nem üres halmaz, a belső állapotok halmaza
- Σ véges, nem üres halmaz, a bemeneti ábécé
- δ: $Q \times \Sigma \rightarrow Q$ az átmenetfüggvény
- $q_0 \in Q$ a kezdőállapot
- $F \subseteq Q$ a végállapotok halmaza

A véges automaták a legegyszerűbb automaták. Véges ábécével, belső állapotokkal rendelkeznek, a kezdőállapotból indulnak és az átmenetfüggvény értelmében minden beolvasott szimbólumra felvesznek egy új állapotot. Ha nincs több beolvasható szimbólum, és az automata végállapotban van, akkor azt mondjuk, hogy felismerte a beolvasott szót, ha nincs végállapotban, akkor az automata nem ismerte fel a beolvasott szót.

2. definíció: Egy $A = (Q, \Sigma, \delta, q_0, F)$ véges automata *konfigurációja* egy (α, q) formális kettes, ahol α az input szalagon még hátra levő szó, q pedig az aktuális belső állapot. Az automata induláskori konfigurációja (ω, q_0), ahol ω a teljes input szó. Az automata feldolgozás közbeni köztes konfigurációja valamely (α, q). Normál működés záró konfigurációja (ε, q'), ahol ε az üres szó, $q' \in F$.

3. definíció: Egy $A = (Q, \Sigma, \delta, q_0, F)$ véges automata egy ω input szót *elfogad* (felismer), ha létezik olyan lépéssorozat, amelynek során a δ leképezés véges sokszori alkalmazása révén az automata induló konfigurációja (ω, q_0) *átvihető* az (ε, q') záró konfigurációba, és $q' \in F$. Ellenkező esetben az automata az input szót *elutasítja*.

A véges determinisztikus automaták a reguláris nyelvek osztályát (az ún. Chomsky 3-as nyelvosztályt) ismerik fel.

A környezetfüggetlen nyelvek (Chomsky 2) felismerésére *veremautomaták* szükségesek.

4. definíció: Egy veremautomata az $M = (Q, V, W, \delta, q_0, z_0, F)$ rendezett hetes, ahol:

- Q a belső állapotok halmaza
- V a bemeneti ábécé
- W a veremábécé
- δ az átmenetfüggvény: $\delta: Q \times (V \cup \{\varepsilon\}) \times W \rightarrow P(Q \times W^*)$
- $q_0 \in Q$ a kezdőállapot
- $z_0 \in W$ a kezdőjel
- $F \subseteq Q$ a végállapotok halmaza

Induláskor az automata a q_0 kezdő állapotban van. A veremben csak egy jel van, a z_0, az input szalagon az input szó jelei vannak felírva, balról-jobbra feltöltve, folytonosan. Az olvasó fej vagy olvas az input szalagról, vagy nem a veremből mindig olvassa a verem tetején levő jelet ezen „input" jelek, és az aktuális belső állapot ismeretében, a δ függvény szerint újabb belső állapotba vált, és a verembe egy szót ír. Az automata megáll, ha a szalagról elfogy az input szó (minden szimbólumot beolvastunk). Az adott szót a veremeautomaták kétféleképpen ismerhetik fel: vagy *végállapottal* (ha az automata végállapotba került), vagy *üres veremmel* (ha az automata verme kiürült).

2. SAJÁT AUTOMATÁK

A következőkben saját automatákat, automatarendszereket vezetünk be, amelyekkel sokkal hatékonyabban elvégezhetők az elemzések, beépíthetjük a szemantikai elemzést, valamint ezek a rendszerek könnyebben implementálhatók.

2.1. Számlálóveremmel rendelkező automaták

A veremautomata mintájára egy egyszerűbb automatát vezetünk be, a *számlálóveremmel rendelkező automatát*, amelynek segítségével egyszerűen megoldható a rekurzív szabályok ellenőrzése, például a (,) zárólejpárok, begin, end stb. párosok ellenőrzése.

5. definíció: Egy számlálóveremmel rendelkező determinisztikus automata az $A = (Q, \Sigma, \delta, q_0, F, M, \Pi)$ rendezett hetes, ahol:

- Q véges, nem üres halmaz, a belső állapotok halmaza
- Σ véges, nem üres halmaz, a bemeneti ábécé
- $\delta: Q \times (\Sigma \cup \{\varepsilon\}) \rightarrow Q \times M \times \Pi$ az átmenetfüggvény
- $q_0 \in Q$ a kezdőállapot
- $F \subseteq Q$ a végállapotok halmaza
- $M = \{Push, Pop, \varepsilon\}$, a veremműveletek halmaza
- Π a veremábéce

A számlálóveremmel rendelkező automatát nemdeterminisztikus alakban is értelmezhetjük a következőképpen:

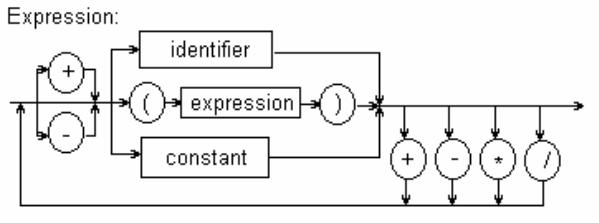
6. definíció: Egy számlálóveremmel rendelkező nemdeterminisztikus automata az $A = (Q, \Sigma, \delta, Q_0, F, M, \Pi)$ rendezett hetes, ahol:

- Q véges, nem üres halmaz, a belső állapotok halmaza
- Σ véges, nem üres halmaz, a bemeneti ábécé
- $\delta: Q \times (\Sigma \cup \{\varepsilon\}) \rightarrow P(Q) \times M \times \Pi$ az átmenetfüggvény
- $Q_0 \subseteq Q$ a kezdőállapotok halmaza
- $F \subseteq Q$ a végállapotok halmaza
- $M = \{Push, Pop, \varepsilon\}$, a veremműveletek halmaza
- Π a veremábéce

Az automata működése megegyezik a véges automata működésével, annyi különbséggel, hogy az automata – átmenetkor – végrehajtja a megfelelő veremműveletet a megfelelő veremábécé jelen. A beolvasott input szó végén, ha az automata verme nem üres, akkor az automata elutasítja az imput szót.

Példa

A következő számlálóveremmel rendelkező automata az aritmetikai kifejezések helyességét ellenőrzi. Az aritmetikai kifejezések szintaxisát a 1. ábrán látható szintaxisdiagram adja meg.



1. ábra. *Aritmetikai kifejezések szintaxisa*

Az 1. ábrán látható diagrammal megadott kifejezéseket az A_1 automata ismeri fel:

$$A_1 = (\{0, 1, 2, 3\}, \{a, k, ;, +, -, *, /,), (\}, \delta, \{0\}, \{3\}, \{Push, Pop, Nop\}, \{(\}),$$

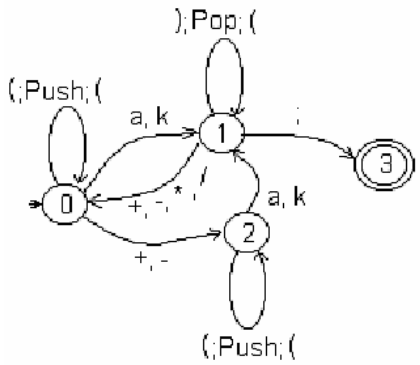
ahol *a* egy tetszőleges változót (azonosítót), *k* pedig egy tetszőleges konstansot jelent.

Az A_1 automata megadható az 1. táblázat segítségével, vagy a 2. ábrán látható gráf segítségével.

Σ	a			k			;			+			-		
Q	**P*(Q)***	***M***	**Π***	**P*(Q)***	***M***	**Π***	**P*(Q)***	***M***	**Π***	**P*(Q)***	***M***	**Π***	**P*(Q)***	***M***	**Π***
0	{1}	*Nop*	ε	{1}	*Nop*	ε	∅	*Nop*	ε	{2}	*Nop*	ε	{2}	*Nop*	ε
1	∅	*Nop*	ε	∅	*Nop*	ε	{3}	*Nop*	ε	{0}	*Nop*	ε	{0}	*Nop*	ε
2	{1}	*Nop*	ε	{1}	*Nop*	ε	∅	*Nop*	ε	∅	*Nop*	ε	∅	*Nop*	ε
3	∅	*Nop*	ε	∅	*Nop*	ε	∅	*Nop*	ε	∅	*Nop*	ε	∅	*Nop*	ε

Σ	*			/			)			(			ε		
Q	**P*(Q)***	***M***	**Π***	**P*(Q)***	***M***	**Π***	**P*(Q)***	***M***	**Π***	**P*(Q)***	***M***	**Π***	**P*(Q)***	***M***	**Π***
0	∅	*Nop*	ε	∅	*Nop*	ε	∅	*Nop*	ε	{0}	*Push*	(	∅	*Nop*	ε
1	{0}	*Nop*	ε	{0}	*Nop*	ε	{1}	*Pop*	(	∅	*Nop*	ε	∅	*Nop*	ε
2	∅	*Nop*	ε	∅	*Nop*	ε	∅	*Nop*	ε	{2}	*Push*	(	∅	*Nop*	ε
3	∅	*Nop*	ε	∅	*Nop*	ε	∅	*Nop*	ε	∅	*Nop*	ε	∅	*Nop*	ε

1. tábázat. *az A_1 automata*



2. ábra. *Az A_1 automata gráf segítségével ábrázolva*

Ha gráf segítségével szeretnénk ábrázolni az automatát, akkor egy irányított multigráfot kell választanunk, amelynek élei és csomópontjai címkézve lesznek. A gráfot a következőképpen építhetjük fel:

- A csomópontokat az állapotokkal címkézzük meg (*Q* elemeivel).
- Ha *(p, Op, z)* ∈ *δ(q, a)*, ahol *q, p* ∈ *Q*, *a* ∈ (Σ ∪ {ε}), *Op* ∈ *M*, *z* ∈ (Π ∪ {ε}), akkor berajzoljuk a *(p, q)* élet, amelyet megcímkézünk az *a; Op; z* sorozattal, vagy ha *Op* = *Nop*, vagy z = ε, akkor csak *a*-val. Ha

egy *p* állapotból több szimbólummal is át lehet menni egy *q* állapotba, akkor ezek a szimbólumok mind felírhatók ugyanarra az élre, vesszővel elválasztva.
- A végállapotokat duplán karikázzuk be.
- A kezdőállapotokhoz bemenő nyilat teszünk.

2.2. A szemantikus elemzés bevezetése, szimbólumok érvényesítése

A szemantikus elemzés, vagy akár a kódgenerálás megvalósítása érdekében *érvényesítő függvényeket* és *kimeneti ábécét* vezetünk be.

7. definíció: Érvényesítő függvénynek nevezünk egy v: $\Sigma \rightarrow \{false, true\}$ Γ követelményrendszer fölött értelmezett függvényt. A v függvény megnézi, hogy a beolvasott szimbólum megfelel-e a követelményrendszernek, ha igen, akkor *true* értéket térít vissza, különben *false* értéket.

A Σ ábécé és Γ követelményrendszer fölött értelmezett érvényesítő függvények halmaza:

$$V = \{v \mid v(s) \in \{false, true\} \ \forall\, s \in \Sigma\}.$$

8. definíció: Egy számlálóveremmel, érvényesítő függvényekkel és kimeneti ábécével rendelkező nemdeterminisztikus automata az $A = (Q, \Sigma, \delta, Q_0, F, M, \Pi, \Gamma, V, \Phi)$ rendezett tízes, ahol:
- Q véges, nem üres halmaz, a belső állapotok halmaza
- Σ véges, nem üres halmaz, a bemeneti ábécé
- $Q_0 \subseteq Q$ a kezdőállapotok halmaza
- $F \subseteq Q$ a végállapotok halmaza
- $M = \{Push, Pop, \varepsilon\}$, a veremműveletek halmaza
- Π a veremábéce
- Γ egy követelményrendszer
- $V = \{v \mid v(s) \in \{false, true\} \ \forall\, s \in \Sigma\}$ az érvényesítő függvények halmaza
- Φ véges, nem üres halmaz, a kimeneti ábécé
- δ: $Q \times (\Sigma \cup \{\varepsilon\}) \rightarrow P(Q) \times M \times \Pi^*) \times V \times \Phi^*$ az átmenetfüggvény

Átmenetkor az automata alkalmazza az érvényesítő függvényt és generálja a kimeneti jelet. Az automata elutasítja a bemeneti szót, ha valamelyik érvényesítő függvény *false* értékkel tér vissza.

3. AUTOMATA RENDSZEREK

Egy adott programozási nyelvhez hozzárendelni egyetlen automatát, amely elemezni tudja a programokat, nehéz, komplikált feladat. Ezt a vállalkozást úgy tudjuk leegyszerűsíteni, hogy a nagy automatát több kisebb automatára bontjuk, majd ezeket összekapcsoljuk. Így *automata rendszerek* jönnek létre.

9. definíció: Egy számlálóveremmel, érvényesítő függvényekkel és kimeneti ábécével rendelkező nemdeterminisztikus automata rendszer az $A = (Aut, Q, \Sigma, \delta, Q_0, F, M, \Pi, \Gamma, V, \Phi)$ rendezett tizenegyes, ahol:
- Aut véges, nem üres halmaz, az automaták halmaza
- Q véges, nem üres halmaz, a belső állapotok halmaza
- Σ véges, nem üres halmaz, a bemeneti ábécé
- $Q_0 \subseteq Q$ a kezdőállapotok halmaza
- $F \subseteq Q$ a végállapotok halmaza
- $M = \{Push, Pop, \varepsilon\}$, a veremműveletek halmaza
- Π a veremábéce
- Γ egy követelményrendszer
- $V = \{v \mid v(s) \in \{false, true\} \ \forall\, s \in \Sigma\}$ az érvényesítő függvények halmaza
- Φ véges, nem üres halmaz, a kimeneti ábécé
- δ: $Aut \times Q \times (\Sigma \cup \{\varepsilon\}) \rightarrow Aut \times P(Q) \times M \times \Pi^*) \times V \times \Phi^*$ az átmenetfüggvény

Ha egy automata befejezte működését, visszatér az őt meghívó automatához. Ez a visszatérés üres veremmel kell hogy megtörténjen, az automata csak így ismeri fel a beolvasott szót.

3.1. Példa – Az *Expr2* nyelv

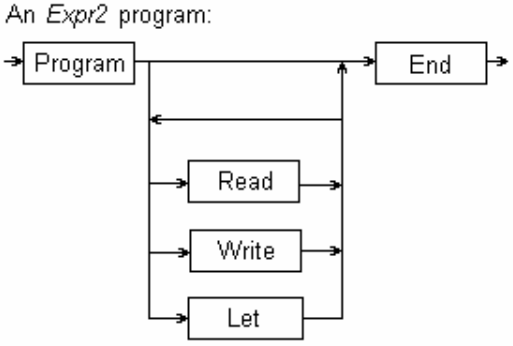
Az *Expr2* nyelv a magas szintű nyelvek osztályába tartozik és kifejezések kiértékelésére alkalmas. A nyelv automatákkal megvalósított fordítóprogrammal rendelkezik.

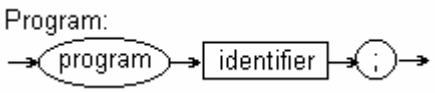
A fordítóprogram forrásállományokat fordít gépi kódra (**.COM** állomány) és megírja az *assembly* nyelvű program-szöveget is, tehát átalakítóként (translator) is működik. A kifejezéseket fordított lengyel ábrázolásmódra alakítja (postfix kifejezések) bináris fák segítségével, és verem használatával megírja a hozzájuk tartozó, futásidejű (run-time) kiértékelő eljárásokat. Az utasítások lexikális, szintaktikai és szemantikai helyességének ellenőrzéséhez a 9. definícióban megadott automatarendszert használja. A bináris fák felépítését és a kódgenerálást az érvényesítő függvények végzik.

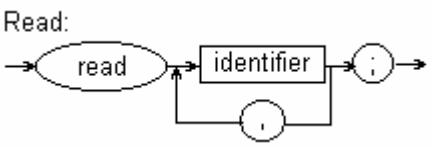
A nyelv relatívan kevés utasítást tartalmaz, mivel speciálisan, kifejezések kiértékelésére volt definiálva. A nyelv utasításkészlete a következő:

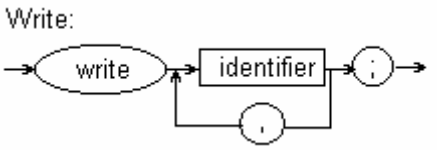
program <azonosító> **;** - a program fejléce.
read <azonosító> {**,** <azonosító>} **;** - beolvasó eljárás.
write <azonosító> {**,** <azonosító>} **;** - kiíró eljerás.
let <azonosító> **:=** <kifejezés> **;** - értékadó utasítás.
end . - a program végét jelöli.

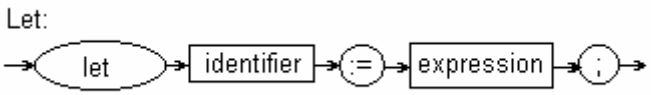
A nyelvet a következő szintaxisdiagramokkal lehet szemléletesebben leírni:

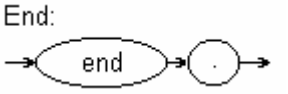


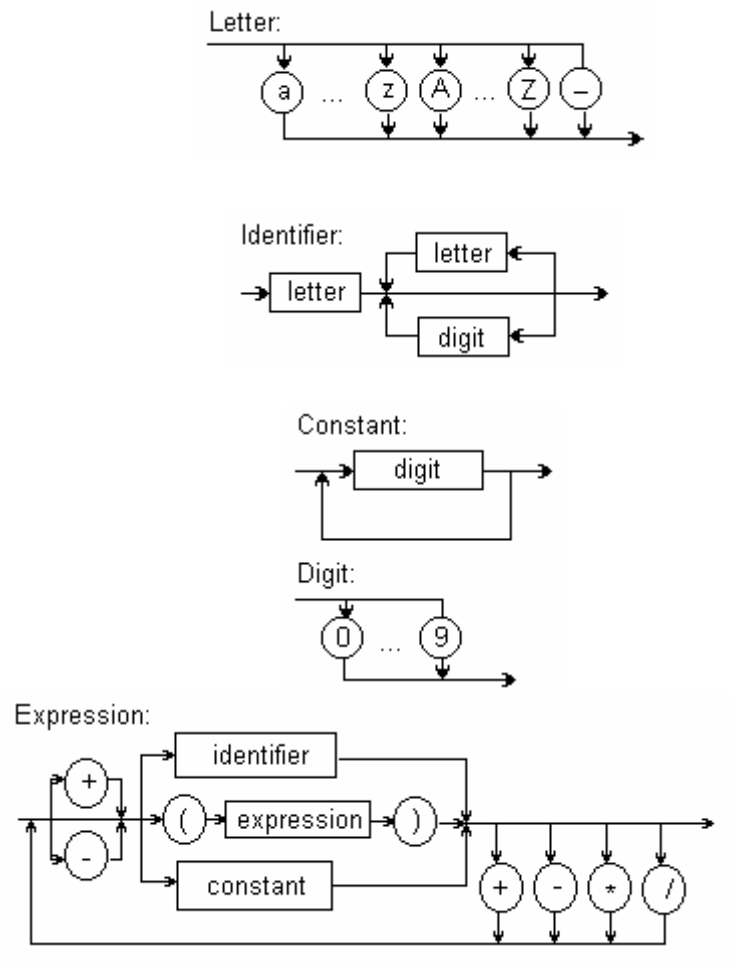












3. ábra. *Az Expr2 nyelv szintaxisdiagramjai*

A szintaxis EBNF leírása:

```
identifier = letter { letter | digit } ;
letter = "a" | "b" | "c" | "d" | "e" | "f" | "g" | "h" | "i" | "j" | "k" | "l" | "m" |
         "n" | "o" | "p" | "q" | "r" | "s" | "t" | "u" | "v" | "w" | "x" | "y" | "z" |
         "A" | "B" | "C" | "D" | "E" | "F" | "G" | "H" | "I" | "J" | "K" | "L" | "M" |
         "N" | "O" | "P" | "Q" | "R" | "S" | "T" | "U" | "V" | "W" | "X" | "Y" | "Z" | "_" ;
digit = "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9" ;
expression = ["+"|"-"] identifier | constant |(expression) ["+" | "-" | "*" | "/" expression] ;
constant = digit { digit }[ "." digit { digit } ] ;
program = "program" identifier ";" [ block ] "end" "." ;
block = "read" | "write" | "let" ;
read = "read" identifier {, identifier} ";" ;
write = "write" identifier {, identifier} ";" ;
let = "let" identifier ":=" expression ";" ;
```

A nyelv egyetlen típusú adatot tud kezelni, ez az egész típus (integer). A változókat nem kell külön deklarálni, értéket értékadáskor vagy beolvasáskor kapnak. A konstansok egész számok. A változókra Pascal-típusú azonosítókkal hivatkozhatunk. Az adatokat 2 byte hosszúságon ábrázoljuk.

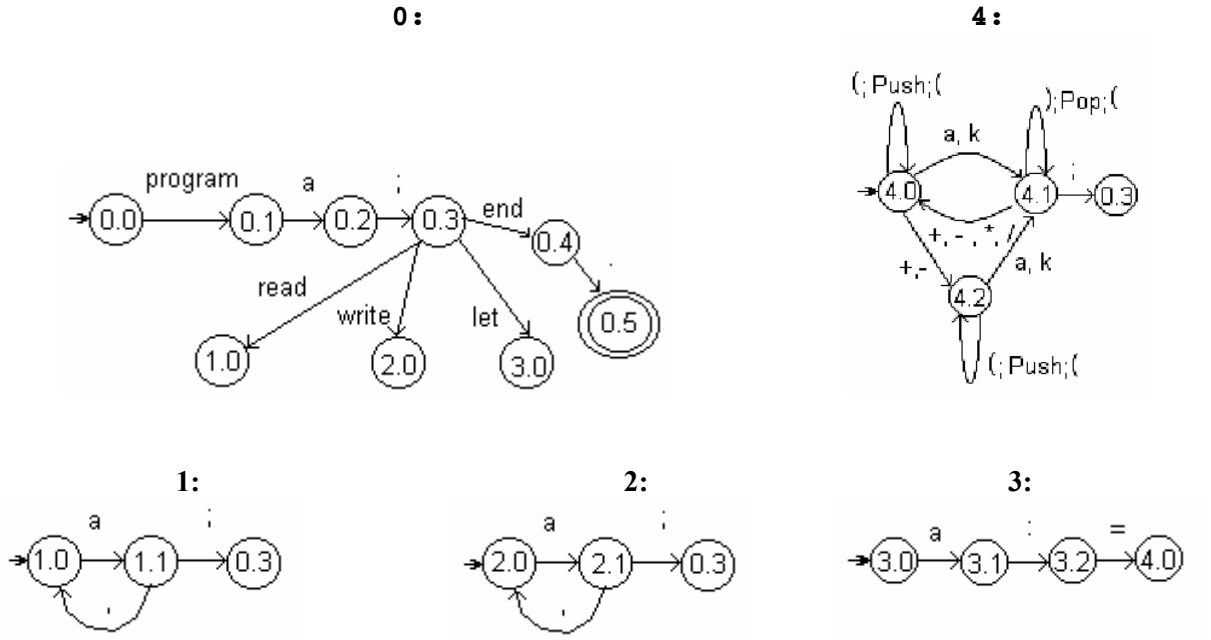
A fordítóprogram a forrásállományt karakterenként olvassa be (tehát egydimenziós fordító), ezekből a lexikális elemző szimbólumokat rak össze. A szimbólumokat fehér karakterek (Space, Tab, Enter) vagy speciális, elválasztókarakterek határolják. Az elválasztókarakterek a következők: +, -, *, /, (,), :, =, amelyek megfelelnek a műveleti jeleknek.

A program fejléccel kezdődik és végszimbólummal fejeződik be. A két rész között az utasítások bármilyen, a szintaktika által megengedett kombinációban szerepelhetnek. Az adatokkal beolvasó, kiíró és értékadó műveleteket végezhetünk. A programokat szöveg-állományokban tároljuk. Egy állomány egy programot tartalmazhat és ezeket az állományokat dolgozza fel a fordítóprogram.

A következő példa egy helyes Expr2 program:

```
program proba;
   read alpha,beta,gamma;
   let a := alpha *+ 50 + (beta * gamma) - 10;
   let alpha := beta -- gamma;
   write a,alpha;
end.
```

A fordításhoz öt automatát használunk. Ezeket kapcsoljuk össze, megfelelő módon, egyetlen egész rendszerré. A használt automatarendszer a következő:



3. ábra. *Az Expr2 nyelvet felismerő automata rendszer*

Az érvényesítő függvények:

Átmenet	*Függvény*
$0.1 \rightarrow 0.2$	*true*, ha az *a* azonosító helyes, és beírja *a*-t a szimbólumtáblába
$1.0 \rightarrow 1.1$	*true*, ha az *a* azonosító helyes, és beírja *a*-t a szimbólumtáblába
$2.0 \rightarrow 2.1$	*true*, ha az *a* azonosító helyes, és ha *a* benne van a szimbólumtáblában
$3.0 \rightarrow 3.1$	*true*, ha az *a* azonosító helyes, és beírja *a*-t a szimbólumtáblába
$4.0 \rightarrow 4.1$ $4.2 \rightarrow 4.1$	*true*, ha az *a* azonosító helyes, ha a *k* konstans helyes, mindketten benne vannak a szimbólumtáblában, ellenőrzi a típusúk kompatibilitását
$4.1 \rightarrow 4.0$ $4.0 \rightarrow 4.2$	*true*, ha a műveletek kompatibilisek *a* és *k* típusaival

2. tábázat. *az Expr2 nyelv érvényesítő függvényei*

4. EKVIVALENCIA A KLASSZIKUS RENDSZEREKKEL

A következő két tétel segítségével bebizonyítjuk, hogy a bevezetett automaták ekvivalensek a klasszikus rendszerekkel, így ezek is a környezetfüggetlen nyelveket ismerik fel.

1. Tétel. *Ekvivalencia a veremautomatákkal*

Bármely $A = (Q, \Sigma, \delta, Q_0, F, M, \Pi)$ számlálóveremmel rendelkező automatához tudunk szerkeszteni egy $A' = (Z, K, T, \delta', z_0, q_0, H)$ veremautomatát úgy, hogy $L(A) = L(A')$.

Bizonyítás

Legyen $A = (Q, \Sigma, \delta, Q_0, F, M, \Pi)$ – megszerkesztjük az $A' = (Z, K, T, \delta', z_0, q_0, H)$ automatát a következőképpen:

$Z = \Pi$; $K = Q \cup q_0$; $T = \Sigma$; $z_0 = \varepsilon$; $H = F$.

Az átmenetfüggvény (δ') a következőképpen adható meg:

- $\delta'(q_0, \varepsilon, z_0) = \{(q, z_0)\}$ bármely $q \in Q_0$
- Legyen $\delta(q, a) = (\{p\}, o, x)$, ahol $q, p \in Q$, $a \in \Sigma \cup \{\varepsilon\}$, $o \in M$, $x \in \Pi^*$, ha $o = Push$, akkor $\delta'(q, a, z) = \{(p, x + z)\}$, ahol $z \in Z^*$ a veremben lévő szekvencia.
- Legyen $\delta(q, a) = (\{p\}, o, x)$, ahol $q, p \in Q$, $a \in \Sigma \cup \{\varepsilon\}$, $o \in M$, $x \in \Pi^*$, ha $o = Pop$, akkor $\delta'(q, a, z) = \{(p, delete(z, x)\}$, ahol $z \in Z^*$ a veremben lévő szekvencia.

Az ekvivalencia bebizonyításához a [6.]-ban bevezetett 2.2. értelmezést és két automata ekvivalenciájának algoritmusát használjuk.

2. Tétel. *Az érvényesítő függvények ekvivalenciája a szemantikus rutinokkal*

A 7. definícióban bevezetett érvényesítő függvények ekvivalensek a [4.]-ben bevezetett szemantikus rutinokkal, az atribútumgrammatikák pedig a 8. definícióban bevezetett automaták alrendszerei.

Bizonyítás

Az 1. tétel, valamint [1.] grammatikák és automaták megfeleltetési tétele alapján belátható, hogy a számlálóveremmel rendelkező automaták ekvivalensek a környezetfüggetlen grammatikákkal. Az attribútumgrammatikákhoz pedig: az érvényesítő függvények a szemantikus rutinok, az attribútumok a függvények paraméterei (bemeneti, kimeneti paraméterek), valamint $\Gamma = C \cup R$.

Az automata rendszer kódgenerálásra is képes, tehát nagyobb rendszer mint az attribútumgrammatikák rendszere.

5. HIBAKEZELÉS

A rendszerben a következő fordítási hibák léphetnek fel:

- Az automatarendszer elakad
- Egy érvényesítő függvény *false* értékkel tér vissza
- Az automatarendszer nem ér végállapotba
- A verem nem ürül ki
- Üres veremből akar elemet kivenni

A lexikális elemző minden szimbólum számára elmenti az előfordulási helyet (sor, oszlop a forrásszövegben), minden automata a rendszerből egy egyedi azonosítóval rendelkezik, hasonlóan minden állapotot be lehet egyértelműen azonosítani.

A rendszer bármely állapotát azonosítani lehet az *(AutomataAzonosító, ÁllapotAzonosító)* párral, a szimbólumot pedig a *(Sor, Oszlop)* párral.

Az automata rendszer állapotából összetett hibaüzenetet tudunk megfogalmazni: a hibán kívül a folytatást is meg tudjuk adni, javasolni tudjuk a helyes szekvenciát.

KÖNYVÉSZET

[1.] Révész György: *Bevezetés a formális nyelvek elméletébe*, Akadémiai Kiadó, Budapest, 1979.

[2.] Aho, A.V.; Ullmann, J.D.: *The Theory of Parsing, Translation and Compiling*, Vol. I.-II. Pretice Hall Inc., Englewood Cliffs, N. J., 1972–1973.

[3.] Aho, A.V.; Ullmann, J.D.: *Principles of Compiler Design*, Addison-Wesley Publishing Co., Reading, Mass., 1977.

[4.] Csörnyei Zoltán: *Bevezetés a fordítóprogramok elméletébe*, Vol. I.-II. ELTE jegyzet, Budapest, 1996.

[5.] Fülöp Zoltán: *Formális nyelvek és szintaktikus elemzésük*, Polygon, Szeged, 1999.

[6.] Kása Zoltán: *Formális nyelvek és automaták*, BBTE, Kolozsvár, 2004.

A backtracking programozási módszer tanítása

Teaching the Backtracking Programming Method

Predarea metodei de programare backtracking

Drd. MARCHIŞ Julianna

Babeş-Bolyai Tudományegyetem, Kolozsvár
marchis_julianna@yahoo.com

ABSTRACT

Understanding and applying the backtracking programming method is difficult for high-school students. But if this method is "discovered" by the student, he/she can apply it easily in problem solving. This is the reason that it is recommended to use the guided discovering method for teaching this topic. In this article there is presented a way to teach backtracking by guided discovering method.

Kulcsszavak: Backtracking, irányított felfedezés

A BACKTRACKING TANÍTÁSA

A tanulók nagy része nehezen, vagy egyáltalán nem érti meg a backtracking programozási módszert, így megpróbálja bemagolni az algoritmust. Ez általában kudarccal jár, mert alkalmazáskor gyakran előfordul, hogy kihagy egy-két utasítást, és az algoritmus már nem lesz helyes; vagy nem tudja felírni, például, a folytathatósági feltételt, és a bemagolt példában levő feltételeket használja. Több osztály eredményeit megfigyelve láthatjuk, hogy a backtracking módszerből írt felmérésen általában gyengébb az osztályátlag, mint a tananyag előző témái esetén. Sajnos vannak tanárok, akik úgy tanítják ezt a módszert, hogy felírják a táblára a kész programot, a tanulók pedig bemásolják a számítógépbe.

A backtracking módszer tanításánál jól lehet alkalmazni az irányított felfedeztetés módszerét. Megfelelően megtervezve és irányítva a tevékenységet, a tanulók lépésről-lépésre fel tudják fedezni a backtracking módszer lényegét, majd az algoritmust. A következőkben ezt szeretném szemléltetni, bizonyos mozzanatokat részletesen leírva, másokat csak megemlítve. A részletesen leírt mozzanatok fontosak ahhoz, hogy a tanulók megértsék a módszert, ezért is fektettem nagyobb hangsúlyt ezekre.

A téma iránti érdeklődés felkeltésére a legalkalmasabb egy játék. Az Internetről ingyenesen letölthető oktatóprogramokban találhatunk ilyen játékot:

– *Királynők elhelyezése a sakktáblán*: Meg kell találni az összes lehetőséget, ahogy elhelyezhetünk 4 királynőt egy 4×4-es sakktáblán úgy, hogy ne üssék ki egymást. Mivel minden megoldást meg kell találni, rendszeresen kell sorra venni az eseteket. Ha a tanulók az első sor első oszlopába helyezik el az első királynőt, akkor a második sor első és második oszlopába nem tehetnek királynőt, így a harmadik oszlopba helyezik. Ekkor a harmadik sorba akárhova is tennék a királynőt, üti az előzőket. Így vissza kell lépni a második sorba, és itt helyezni máshova a királynőt – a negyedik oszlopba. Újra a harmadik sorba lépünk, és itt próbáljuk elhelyezni a királynőt, stb. Így végigvéve az eseteket, a tanulók megértik a módszer lényegét. Viszont a játékot úgy is lehet játszani, hogy nem az első sorral kezdjük, hanem például először a második sorba, majd a negyedik sorba, stb. Ha a tanulók így járnak el, akkor a játék nem segít a backtracking módszer lényegének felfedezésében.

– *Egy házaló által bejárt út*: Egy házaló házról házra járva árulja termékeit. Bizonyos házak között van direkt út, mások között nincs. Meg vannak rajzolva a házak és az utak, segíteni kell a házalónak megtalálni azt az útvonalat, amelyen minden házhoz eljut, visszaérve a kiinduló ponthoz, és minden úton csak egyszer halad el. Ez a játék is nagyon jól tükrözi azt, hogy ha rájövünk arra, hogy egy adott háztól nem léphetünk tovább, vissza kell menni az előző házhoz, és onnan keresni más utat.

– *Labirintus*: Egy labirintuson kell végighaladni. Itt is, ha zsákutcába jutunk, vissza kell térni.

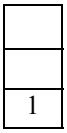
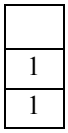
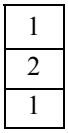
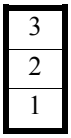
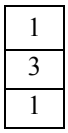
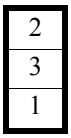
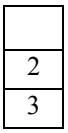
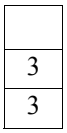
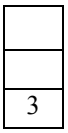
A fenn felsorolt valamely játék segítségével a tanulók megértik a backtracking módszer alapgondolatát, mint ahogy a neve is mondja: a visszalépéses keresés módszere.

Egy példa segítségével fel lehet fedezni a módszert lépésről lépésre. Ehhez egy olyan példát kell választani, amelyet könnyű végigvezetni, és minden esetet tárgyalni.

Vegyük a permutációk generálását: Írjuk fel az {1, 2, 3} halmaz permutációit.

A megoldások: {1, 2, 3}, {1, 3, 2}, {2, 1, 3}, {2, 3, 1}, {3, 2, 1}, {3, 1, 2}.

Észrevehetjük, hogy egy-egy megoldást tömbben tárolhatunk. Az előző játékokból láthatták a tanulók, hogy először a tömb első elemét keressük meg, tehát felfoghatjuk veremként. A verem minden szintjére az 1, 2 vagy 3 számok valamelyike kerül. Azt is észrevehetjük, hogy a megoldások különböző elemekből állnak. Azt a feltételt, amelyet a megoldás elemei teljesítenek, belső feltételnek nevezzük. Ezért, ha például az első és a második szinten levő elem megegyezik, akkor nem érdemes a harmadik szintre menni, és oda tenni elemet, mert úgysem kapunk megoldást. Azt a feltételt, amely kell teljesüljön ahhoz, hogy érdemes legyen a megoldást tovább generálni, folytathatósági feltételnek nevezzük. Azt is észrevesszük, hogy például az {1, 2} halmaz esetén teljesül ugyan a folytathatósági feltétel, de nem megoldás. Ahhoz, hogy megoldás legyen, 3 eleme kell legyen a halmaznak. Hasznos, ha végigvezetjük a tanulókkal a megoldás keresését az alább leírt módon. Itt csak az elejét és a végét adtuk meg ennek a levezetésnek, de az órán minden esetet tárgyalnánk.

						
– az I. szintre betesszük az első elemet – teljesül a folytathatósági feltétel, a következő szintre lépünk	– a II. szintre betesszük az 1-et – nem teljesül a folytathatósági feltétel, új elemet teszünk a II. szintre	– a II. szintre betesszük a 2-t – teljesül a folytathatósági feltétel, de nem megoldás: továbblépünk a következő szintre	– a III. szintre betesszük az 1-et – nem teljesül a folytathatósági feltétel, új elemet teszünk a III. szintre	– a III. szintre betesszük a 2-t – nem teljesül a folytathatósági feltétel, új elemet teszünk a III. szintre	– a III. szintre betesszük a 3-at – teljesül a folytathatósági feltétel – megoldás	– próbálunk más elemet tenni a III. szintre, de minden lehetőséget kipróbáltunk, ezért visszalépünk az előző szintre, és ott folytatjuk
			...			
– a II. szintre betesszük a 3-at – teljesül a folytathatósági feltétel, nem megoldás: továbblépünk	– a III. szintre betesszük az 1-et –nem teljesül a folytathatósági feltétel, új elemet teszünk a III. szintre	– a III. szintre betesszük az 2-t – teljesül a folytathatósági feltétel – megoldás	...	... – visszaléptünk – új elemet teszünk a II. szintre	– a II. szintre betesszük a 3-at – nem teljesül a folytathatósági feltétel, új elemet teszünk a II. szintre –minden elemet kipróbáltunk: visszalépünk	– az első szintre kellene más elemet tenni, de már minden lehetőséget kipróbáltunk, visszalépünk

A fenti lépéseket lehet a táblára írni, vagy egy számítógépes program segítségével végigvezetni, de fontos az, hogy minden lépést megindokoljunk. Lehet, hogy az elején az indoklást a tanár kell végezze, de majd feltétlenül kapcsolódjanak be a tanulók is, diktálva a tanárnak a lépéseket. Végigvezetve a megoldás generálásának lépéseit, a tanulók levonhatják a következtetéseket: milyen típusú feladatok esetén alkalmazható a módszer, mi a belső feltétel, mi a folytathatósági feltétel, mikor kapunk egy megoldást és melyek a módszer lépései. Ezekre a tanár kérdések segítségével vezeti rá a tanulókat, és fontos az, hogy a füzetbe is leírják.

A továbbiakban az algoritmust szeretnénk feliratni a tanulókkal. A fenn leírtak alapján láthatjuk, hogy szükségünk lesz három függvényre:

– *próbál* – próbál tenni egy elemet egy adott szintre (tehát a szint paramétere lesz). Ha sikerül tennie igazat ad vissza, különben hamisat.

– *folytat* – ellenőrzi a folytathatósági feltételt egy adott szintre.

– *megoldás* – ellenőrzi, hogy megoldás-e.

Most térjünk vissza a fenti megoldás során felfedezett lépésekhez:

– a megoldás generálásakor az első szintről indulunk, ide betesszük az első elemet.

– általában minden szinten a következőket végezzük el: próbálunk tenni egy elemet. Ha tudunk tenni új elemet, akkor ellenőrizzük, ha teljesül a folytathatósági feltétel. Ha teljesül, akkor ellenőrizzük, ha megoldást kaptunk-e. Ha igen, akkor kiíratjuk, ha nem, akkor továbblépünk a következő szintre. Ha nem teljesül a folytathatósági feltétel, akkor új elemet próbálunk tenni az adott szintre. Ha nem sikerül új elemet tenni, akkor visszalépünk az előző szintre.

– az első szintről indulva és ismételve az algoritmust, amíg minden szinten minden elemet kipróbáltunk, megkapjuk az összes megoldást.

Fontos, hogy az algoritmust lépésről lépésre írjuk fel, egészítsük ki. Írhatjuk a táblára, figyelmeztetve a diákokat, hogy még ne írják, mert folyamatosan egészítjük ki, esetleg javítjuk ki.

– egy szinten próbálok tenni (*sz*-el jelölve a szintet), ha sikerül tenni, akkor ellenőrzöm, hogy teljesül-e a folytathatósági feltétel:

```
tett := probal(sz);
if tett then folyt := folytat(sz);
```

– ezt ismételni kell. Addig próbálok új elemet tenni egy adott szintre, amíg teljesül a folytathatósági feltétel, vagy már nincs kipróbálatlan elem:

```
repeat
  tett := probal(sz);
  if tett then folyt := folytat(sz);
until (not tett) or (tett and folytat);
```

– két esetben lépünk ki a repeat-ből. Ha azért léptünk ki, mert tettünk elemet és teljesül a folytathatósági feltétel, akkor ellenőrizzük, hogy megoldást kaptunk-e. Ha igen, akkor kiíratjuk. Ha nem megoldás, akkor a következő szintre lépünk, és inicializáljuk a szint értékét. Ha azért léptünk ki, mert az adott szinten minden elemet kipróbáltunk, akkor visszalépünk az előző szintre:

```
if tett and folytat then
  if megoldas(sz) then kiir(sz)
                  else
                    begin
                      inc(sz);
                      v[sz] := 0;
                    end;
                    else dec(sz);
```

– mindezt addig ismételjük, míg minden szinten minden elemet kipróbáltunk, vagyis a 0. szintre lépünk vissza:

```
sz := 1; v[sz] := 0;
while sz > 0 do
  begin
    repeat
      tett := probal(sz);
      if tett then folyt := folytat(sz);
    until (not tett) or (tett and folytat);
    if tett and folytat then
      if megoldas(sz) then kiir(sz)
                      else
                        begin
                          inc(sz);
                          v[sz]:=0;
                        end;
                      else dec(sz);
  end;
```

Ezután megírjuk az algoritmusban használt függvényeket és eljárásokat. A tanulók beírják az egész programot a számítógépbe, és ellenőrzik, hogy működik-e.

Láthattuk, hogy ilyen módon a tanulók írják fel az algoritmust, nem a tanár. Így jobban megértik a módszert. Fontos az, hogy gyakorlás során se kérjük a tanulóktól, hogy egyből felírják az egész algoritmust, mert az arra készteti őket, hogy bemagolják. Minden feladat megoldásánál időt kell biztosítani arra, hogy a tanulók újra felírják, „feltalálják” az algoritmust.

Gyakorlás során minden feladat esetén meg kell beszélni a tanulókkal, hogy hogyan kódoljuk az eredményeket, melyek a belső feltételek és a folytathatósági feltételek, mikor kapunk eredményt. Javasolt a feladatokat csoportosítani:

– olyan feladatok, amelyek esetén az $\{1, 2, \ldots, n\}$ elemekkel dolgozunk. Például írjuk fel az 1, 2, …, n elemek k elemű variációit; írjuk fel az $\{1, 2, \ldots, n\}$ halmaz összes részhalmazát, stb.
– olyan feladatok, amelyek hasonlóak az előzőekhez, de az $\{x_1, x_2, \ldots, x_n\}$ elemekkel dolgozunk. Például: Egy versenyen n sportoló vesz részt, mindegyiknek tudjuk a nevét, az országot, melyet képvisel és a versenyszámát. Egy napon $m < n$ sportoló versenyezhet. Írjuk ki az összes lehetőséget, ahogy sorra kerülhetnek a sportolók, tudva azt, hogy nem lehet egymás után két versenyző ugyanabból az országból, és a versenyszámuk növekvő sorrendjét be kell tartani.
– olyan feladatok, ahol az $\{ x_1, x_2, \ldots, x_n\}$ elemekkel dolgozunk, és ahol a folytathatósági feltételek ellenőrzésére bizonyos értékeket kell kiszámolni. Például ilyen feladat az összegkifizetés adott értékű bankjegyekkel.
– olyan feladatok, ahol egy mátrix is szükséges az adatok tárolására, például a térképszínezés (azt tároljuk a mátrixban, hogy két ország szomszédos-e vagy sem), a házaló árus (azt tároljuk a mátrixban, hogy két ház között van-e direkt út) feladat esetén.

A tanulók is megfogalmazhatnak feladatokat. Az osztályt 3-4 tanulós csoportokra osztjuk, minden csoport szerkeszt egy backtracking módszerrel megoldható feladatot. A feladatot átadják egy másik csoportnak, mely megoldja. A megoldást az a csoport ellenőrzi, amely az illető feladatot javasolta. Ez egy jó gyakorlat arra, hogy a tanulók megtanulják, hogyan kell egy feladatot megfogalmazni, melyek a feladat megoldásához feltétlenül szükséges adatok. A másik csoport megoldását ellenőrizve visszajelzést kapnak arról, hogy jól dolgoztak-e, egyértelműen, érthetően fogalmazták-e meg a feladat szövegét.

Míg a hagyományos tanításban a tanár elmondja az új anyagot, magyaráz, a tanulók meg leírják, próbálják megérteni, és visszaadni a hallottakat; az irányított felfedeztetés módszerével a tanuló fedezi fel az új ismeretet a tanár irányításával. Ehhez viszont szükséges az, hogy a tanulók tudják azokat az ismereteket, amelyekre építeniük kell. Az irányított felfedeztetés során bizonyos fogalmak, ismeretek ismételten előkerülnek, ezáltal rögzülnek. E módszer alkalmazásakor a felfedezés és a gyakorlás szoros kapcsolatban van. Tulajdonképpen az új ismerethez kötődő algoritmus felfedezése hasonló egy adott feladat megoldásához szükséges algoritmus megtalálásához.

KÖNYVÉSZET

[1.] Ionescu Klára: *Metodica predării informaticii*, BBTE, Kolozsvár, 2001.
[2.] George Daniel Mateescu, Pavel Florin Moraru, Otilia Sofron: *Informatică (X.)*, Petrion, Bukarest, 2000.

Egész együtthatós polinomok irreducibilis tényezőkre bontása

Decomposition Of Algebraic Polynomes with Integers Coefficients in Irreducible Factors

Descompunerea polinoamelor cu coeficienți întregi în factori ireductibil

[1]Dr. PÁTER Zoltán, [2]Dr. OLÁH-GÁL Róbert

[1]540479 Marosvásárhely, Înfrățirii utca 1. szám, Telefon: 0265.242812
[2]Sapientia – Erdélyi Magyar Tudományegyetem, Csíkszereda ,ogrogrogr@gmail.com

ABSTRACT

In this study we define a new algorithm for the decomposition of algebraic polynomials with integer coefficients in irreducible factors. This algorithm is simple and only uses elementary operations, but it uses the algorithm for the decomposition of an integer number to prime factors. The main ideas of the algorithm bases on the analogy between the giving of the polynomial and a number in number system with base B.

Kulcszavak: polinomok, irreducibilitás, tényezőre bontás

BEVEZETÉS

Az algoritmikában, az egész együtthatós polinomok irreducibilis tényezőkre bontása épp oly fontos, mint a számok prímtényezőkre bontása és természetesen ismeretelméletileg a két probléma analóg. Knuth referenciakönyvében [1] mind a két problémát részletesen tárgyalja és bemutatja az addig ismert leghatékonyabb algoritmusokat. Ezen algoritmusok között vannak egyszerűek és nagyon nehezek és sok esetben, hosszú és fáradságos munka egy-egy algoritmus kipróbálása.

A fenti algoritmus egyszerű és didaktikai szempontból előnyösebb a bemutatása, mint Knuth könyvében [1] ismertetett Berlekamp vagy Golomb, Welch, Hales algoritmusai, szerintünk a hatékonysága is jó, de ebben nem tudunk határozott véleményt mondani, mert Berlekamp algoritmusát még nem próbáltuk ki. Az algoritmus alapötlete a polinomok és egy szám B alapú számrendszerben való felírásának alakja közötti analógián nyugszik.

TÉNYEZŐKRE BONTÁS

Jelen dolgozatban tanulmányozzuk a polinomok irreducibilis tényezőkre bontását ***Z*** fölött. Tehát minden polinom *Z[X]*-hez tartozik, azaz együtthatói egész számok.

1. Előre bocsátunk egy segédtételt:

Lemma. Ha $a_k, a_{k-1}, ..., a_{k-l}$ természetes számok, $a_k \neq 0$ és $f = a_kX^k - a_{k-1}X^{k-1} - ... - a_{k-l}X^{k-l}$ (1)
akkor:

$$f = (a_k-1)X^k + (X-a_{k-1}-1)X^{k-1} + ... + (X-a_{k-l+1}-1)X^{k-l+1} + (X-a_{k-l})X^{k-l}. \quad (2)$$

Bizonyítható teljes indukcióval l szerint.

2. Legyen $n \in N^*$, B alapú számrendszerben felírva

$$n = b_mB^m + b_{m-1}B^{m-1} + ... + b_1B + b_0. \quad (3)$$

Ekkor n-ben előfordulnak $b_kB^k + b_{k-1}B^{k-1}$ tagok és végezzük el a következő műveletet:
$b_kB^k + b_{k-1}B^{k-1} = b_kB^k + (B - (B - b_{k-1}))B^{k-1} = b_kB^k + B^k - (B - b_{k-1})B^{k-1} = (b_k + 1)B^k - (B - b_{k-1})B^{k-1}$.

Ha $b_k + 1 = B$, akkor $b_kB^k + b_{k-1}B^{k-1} = (b_{k+1} + 1)B^{k+1} - (B-b_{k-1})B^{k-1}$.

Az eljárás folytatódik, ha $b_{k+1} + 1 = B$. Ekkor $n = b_mB^m + ... + (b_k + 1)B^k - (B-b_{k-1})B^{k-1} + ... + b_0$, illetve $n = b_mB^m + ... + (b_{k+1} + 1) B^{k+1} - (B - b_{k-1}) B^{k-1} + ... + b_0$, és így tovább.

A fenti műveletet több együtthatón is elvégezhetjük és n-et különböző alakokban kapjuk meg felírva, azaz

$$n = b_p'B^p + b_{p-1}'B^{p-1} + ... + b_1'B + b_0 \quad (4)$$

alakban, ahol b_i' lehet negatív egész szám is, de mindig $|b_i'| < B$.

A (4) alatti alakot n-nek egy *felépített alakjá*nak nevezzük. Ha (4)-ben B helyett X-et írunk, akkor kapjuk a

$$g = b_p'X^p + b_{p-1}'X^{p-1} + ... + b_1'X + b_0 \quad (5)$$

polinomot, amelyet egy *n-hez felépítés által asszociált* polinomnak nevezünk.

Evidens, hogy egy n ($\in N^*$) számhoz végtelen sok polinom asszociálható felépítés által és mindegyik főegyütthatója pozitív.

3. Legyen n (3) alakban felírva. Ekkor a $b_kB^k + b_{k-1}B^{k-1}$ binommal a következő műveleteket végezhetjük el:
$b_kB^k + b_{k-1}B^{k-1} = (b_k-1) B^k + (B + b_{k-1}) B^{k-1} = (b_k - 2) B^k + (2 B + b_{k-1})B^{k-1} = ...$ stb.

Ezeket a műveleteket mindenik, sőt több együtthatóval egyidejűleg elvégezhetjük és akkor megkapjuk n lebontott alakjait. Ha most (4) n-nek egy lebontott alakja, akkor (5)-t egy *n-hez lebontás által asszociált* polinomnak nevezzük.

4. Legyen most, $f, g, h \in Z[X]$ és

$$f = a_nx^n + ... + a_0,\ g = b_mX^m + ... + b_0,\ h = c_pX^p + ... + c_0 \quad (6)$$

és

$$f = g \cdot h \quad (7)$$

Legyen továbbá B egy olyan természetes szám, amelyik f, g és h mindenik együtthatója moduluszánál nagyobb. Evidens, hogy

$$f(B)=g(B).h(B). \quad (8)$$

Továbbá f, g és h-ban csoportosítjuk a tagokat úgy, hogy minden csoport csak egy variációt tartalmazzon (azaz (1) alakú legyen) és mindenik csoportot írjuk fel (2) alakban. Ekkor (8)-ban $f(B), g(B)$ és $h(B)$ B alapú számrendszerben felírt számok és f, g, h $f(B), g(B)$ illetve $h(B)$-hez felépítés által asszociált polinomok. Tehát f tényezőit $f(B)$ osztóihoz, B alapú számrendszerben asszociált polinomjai között kell keresni.

Illusztráljuk ezt egy példán. Legyen

$$f = X^4-3X^3 + 5X^2 - 4X + 2$$
$$g = X^2 - X + 1,\ \ h = X^2 - 2X + 2.$$

Ekkor $f = g. h$ és írjuk át mindegyiket (2) szerint:

$$f = (X-3) X^3 + 4X^2 + (X-4)X + 2$$
$$g = (X-1)X + 1,\ h = (X-2)X+2$$

és B-nek válasszunk egy 4-nél nagyobb egész számot, például 10-et. Ekkor

$$f(10) = 7\cdot10^3+4\cdot10^2+6\cdot10+2 = 7462$$
$$g(10) = 9\cdot10+1=91,$$
$$h(10) = 8\cdot10+2=82$$
$$7462=91\cdot82$$

Továbbá: $91 = 9\cdot10 + 1= (10-1)\ 10 + 1=10^2 - 10 + 1.$

Az asszociált polinomok $9X+1$ és X^2-X+1. $9X+1$ nem lehet osztója f-nek, mert főegyütthatója nem osztója f főegyütthatójának. Marad mint lehetőség X^2-X+1.

Továbbá: $82 = 8\cdot10 + 2 = (10-2)\cdot10 + 2 = 10^2 - 2.10 + 2$

Az asszociált polinomok $8\ X + 2,\ X^2 - 2X +2$.

Az első elesik, mert 8 nem osztója 1-nek. Tehát mindkét osztó $f(10)$ egy-egy osztójához felépítés által asszociált polinom.

5. A következő algoritmus adható meg egy f polinom felbontására:

(i) választunk egy B pozitív egész számot, amely f mindegyik együtthatója moduluszánál nagyobb. (Célszerű 10 hatványai közül válogatni, mert akkor 10-es számrendszerben számolunk).

(ii) $f(B)$-t törzstényezőkre bontjuk.

(iii) Felírjuk $f(B)$ összes pozitív osztóit B alapú számrendszerben.

(iv) Felírjuk $f(B)$ minden osztójának felépítés által asszociált polinomja közül azokat, melyeknek foka kisebb mint f-nek foka.

(v) Az asszociált polinomok közül kihúzzuk azokat amelyeknek főegyüttható nem osztója *f* főegyütthatójának és azokat, amelyeknek szabadtagja nem osztója *f* szabadtagjának.

(vi) Legyen *g* egy felépítés által asszociált polinomja *f(B)* egy osztójának, amelyik az (v)-ben előírt próbákat kiállta. Válasszunk egy *A* kontrolszámot és azt mind *f*-be mind az összes asszociált polinomokba behelyettesítjük, így *g*-be is. Ha *f(A)* nem osztható *g(A)*-val, akkor *g*-t töröljük.

(vii) Az így maradt polinomokkal elosztjuk *f*-et.

Megjegyzés: Ha $f(B) = m \cdot n$ és *m* egyik felépítés által asszociált polinomja nem osztója *f*-nek, akkor *n* felépítés által asszociált polinomjait fel se kell írnunk, mert nem lehetnek osztók.

6. Példa

$f = X^5 - 3X^4 - 4X^3 + 8X^2 + 10X + 3$

$f(100) = 9696081003 = 3 \cdot 173 \cdot 809 \cdot 3299 \cdot 7$

$f(100)$ osztói: $3, 7, 173, 809, 3299, 3 \cdot 7 = 21, 3 \cdot 173 = 519, 3 \cdot 809 = 2427,$
$3 \cdot 3299 = 9897, 7 \cdot 173 = 1211, 7 \cdot 809 = 5663, 7 \cdot 3299 = 23093, 173 \cdot 809 = 139957,$
$173 \cdot 3299 = 570727$ stb.

Az osztók felsorolását itt abbahagyhatjuk, mert ha az egyenként és kettőként vett prím osztókhoz asszociált polinomok közül egyik sem osztója *f*-nek, akkor a háromként vett prím osztók szorzatához asszociált polinomok nem lehetnek *f* osztói.

Vegyük sorra *f* osztóit. Mindegyik felépített alak alá odaírjuk az asszociált polinomot.

$3 =$	$100 - 93 =$	$100^2 - 99 \cdot 100 - 93 =$	stb.
	$X-97$	$X^2-99 \cdot X-97$	

Mindegyik asszociált polinom szabadtagja *-93*, tehát nem lehet egyik sem osztója *f*- nek.

Tovább menve eljutunk *173*-ig.

$173 =$	$100 + 73 =$	$2 \cdot 100 - 27 =$	$100^2 - 98 \cdot 100 - 7 =$	stb.
	$X+73$	$2 \cdot X-27$	$X^2-98 \cdot X-27$	

egyik sem lehet osztója *f*-nek a *73* illetve a *-27* szabadtag miatt.

Tovább menve eljutunk a *3·3299* osztóhoz:

$3 \cdot 3299 =$	$9897 =$	$98 \cdot 100 + 97 =$	$99 \cdot 100 - 3 =$
		$98 \cdot X+97$	$99X-3$

$100^2 - 100 - 3 =$	$100^3 - 99 \cdot 100^2 - 100 - 3$
$X^2 - X - 3$	$X^3 - 99 \cdot X^2 - X - 3$

Itt az első kettő elesik, a negyedik értéke $X = 1$-re egyenlő -102 és $f(1)=15$. Mivel *15* nem osztható *102*-vel, ez is elesik. Marad esélyesnek az X^2-X-3.

Próbáljuk ki a társosztót:

$7 \cdot 173 \cdot 809 =$	$979699 =$	$97 \cdot 100^2+96 \cdot 100+99 =$	$97 \cdot 100^2+97 \cdot 100-1 =$
		$97 \cdot X^2+96 \cdot X+99$	$97 \cdot X^2+97 \cdot X-1$

$98 \cdot 100^2 - 3 \cdot 100 - 1 =$	$100^3 - 2 \cdot 100^2 - 3 \cdot 100 - 1$
$98 \cdot X^2 - 3 \cdot X - 1$	$X^3 - 2 \cdot X^2 - 3 \cdot X - 1$

Ezek közül esélyes az $X^3 - 2 \cdot X^2 - 3 \cdot X - 1$. Ekkor kipróbálva:

$(X^2 - X - 3)(X^3 - 2X^2 - 3X - 1) = X^5 - 3X^4 - 4X^3 + 8X^2 + 10X + 3 = f.$

Tehát felbontottuk *f* -et.

7. A 4. pontban kifejtettek alapján *B* egy olyan szám kell legyen, hogy (7)-ben, mind *f*, mind *g* , mind *h* minden együtthatója moduluszánál nagyobb legyen. Csakhogy mi csak *f*-et ismerjük, *g*-t, *h*-t nem, tehát nem biztos, hogy a választott *B* szám nagyobb *g* és *h* együtthatói moduluszánál. Ezért ha *f(B)* egyik osztójához felépítéssel asszociált polinom sem osztja *f*-et, még nem lehetünk biztosak, hogy *f* irreducibilis. Ezt az esetet fogjuk most megvizsgálni.

(7)-ben illetve (6)-ban legyen $|b_k| \geq B$.

Ekkor $b_k = B.q + r$, $0 \leq r < B$ és $g(B) = b_m B^m + \ldots + (b_{k+1} + q)B^{k+1} + qB^k + \ldots$

Ha $b_{k+1} + q \geq B$ vagy $b_{k+1} + q < 0$, akkor ismét alkalmazzuk a fenti eljárást és kapjuk *g(B)*-nek egy *B* alapú számrendszerben felírt alakját. *g(B)* eredeti alakja ennek akkor egy lebontott alakja. Mielőtt az

irreducibilitást kijelentenénk, meg kell vizsgáljuk *f(B), g(B)* osztójához lebontás által asszociált polinomokat is.

Még egy probléma merül fel, mivel egy számnak végtelen sok lebontása van, hol állunk meg?

A 3. alapján, lebontás útján:

$g(B) = ... + (b_k - q)B^k + (q \cdot B + b_{k-1})B^{k-1} + ...$ (9)

alakú. Az ehhez az alakhoz asszociált polinom

$$g' = ...+(b_k - q)\, X^k + (q \cdot B + b_{k-1})\, X^{k-1} + ...$$

és $f = g' \cdot h$ esetén $f(1) = g'(1) \cdot h(1)$, illetve $|f(1)| = |g'(1)|\ |h(1)|$. Tehát $|g'(1)| \le |f(1)|$.

De $|g'(1)| = |... + b_k + (q-1)\, B + b_{k-1} + ...|$ és ha q akármilyen nagy lehet, akkor $|g'(1)|$ is korlátlanul nő.

Tehát a lebontásokban csak addig megyünk, míg a lebontás által asszociált polinom együtthatói összegének abszolút értéke kisebb mint *f* együtthatói összegének abszolút értéke.

8. Példa

$$f = X^3 + 3X^2 - 2X - 2.$$

Legyen $B = 4$, $f(B) = 102 = 2 \cdot 3 \cdot 17$

$34 = 2 \cdot 4^2 + 2$ ennek egy lebontása $34 = 4^2 + 4^2 + 2 = 4^2 + 4 \cdot 4 + 2$, és az asszociált polinom:

$X^2 + 4X + 2$. Kipróbálva $f = (X^2 + 4X + 2) \cdot (X - 1)$

Látható, hogy az egyik tényező egyik együtthatójának a modulusza.

(Az olvasó tekintsen el attól, hogy ha felépítéssel megkaptuk volna az $X-1$ osztóit is akkor a lebontásra már nincs szükség).

AZ ALGORITMUS

Az algoritmust mi a következő formában valósítottuk meg, egy IBM-PC-n, Turbo Pascal-ban:

Jelöljük a felbontásra kiválasztott polinomot f-fel.

(i) választunk egy *B* pozitív egész számot, amely f mindegyik együtthatója moduluszánál nagyobb.

(ii) *f(B)*-t törzstényezőkre bontjuk.

(iii) Felírjuk *f(B)* összes pozitív osztóit.

(iv) Vesszük (sorba) *f(B)* pozitív osztóit és felírjuk *B* alapba.

(v) Az így kapott együtthatókat (jelöljük *g*-vel) variálni kezdjük és addig végezzük a variálást amíg az így asszociált polinom nem osztja *f*-et, vagy $|g(1)| > |f(1)|$, vagy $deg(g) > deg(f)$.

(vi) Ha sikerült a felbontás, akkor kiírjuk a *g* együtthatóit és megállunk, ha nem, akkor vesszük a következő osztót és visszatérünk (iv)-re.

(vii) Ha az összes osztót kimerítettük és mégsem sikerült a felbontás, akkor kiírjuk, hogy *f* irreducibilis és megállunk.

Az algoritmus matematikai alapja garantálja a felbontás sikerét, amit az eddigi tesztelés egyértelműen igazolt. Látható, hogy ebben az algoritmusban az együtthatók hatékony variálása a lényeg.

Az együtthatók variálásán azt értjük, hogy ha:

$n = b_m B^m + b_{m-1} B^{m-1} + ... + b_1 B + b_0$ a hozzárendelt polinom pedig:

$g = b_m X^m + b_{m-1} X^{m-1} + ... + b_1 X + b_0$ akkor az együtthatókon a következő csoportosításokat és összevonásokat végezhetjük el.

1. (első fajú variálás): ha $b[k]+1=B$ akkor $b[k+1]=b[k+1]+1$ és $b[k]=-1$.

2. (másod fajú variálás): ha $b[k]+1= B$ akkor $b[k+1]= b[k+1]+1$ és $b[k] = 0$ és $b[k-1] = b[k-1]-B$.

3. (harmad fajú variálás): ha $|b[k+1]+1| < B$ és $|b[k]-B| < B$, akkor $b[k+1]= b[k+1]+1$ és $b[k] = b[k]-B$.

És még elképzelhető egy 4. variálás is, ahol nem kötnénk ki, hogy a variálás a *B* alapban történjen.

De ennek a 4. fajú variálásnak a hiányában is jól dolgozik az algoritmus. Talán a 3.-at is ki lehetne iktatni. Mi a variálást illetően, a kódolás egyszerűsítéséért egy kicsit eltértünk az eredeti dolgozattól, abban a reményben, hogy ez logikailag egyenértékű az ott kifejtettekkel. Természetesen további tanulmányozás alapját képezi a leghatékonyabb variálások kiválasztása és a *B* alap megválasztása. Továbbá nem kellene *f(B)*-t törzstényezőre bontani, csak miután egy osztóval való próbálkozás sikertelen volt, azután kellene egy újabb osztót generálni és azzal tovább folytatni.

Egy első tesztsorozatban, az összes osztó helyett, a prímtényezőket csak kettesével vettem és a következő variánssal dolgoztam.

Jelöljük a felbontásra kiválasztott polinomot *f*-fel.

I. választunk egy *B* pozitív egész számot, amely *f* mindegyik együtthatója moduluszánál nagyobb.

II. *f(B)*-t törzstényezőkre bontjuk.

III. Felírjuk *f(B)* egy és két szorzótényezős pozitív osztóit.
IV. Vesszük (sorba) *f(B)* pozitív osztóit és felírjuk *B* alapba.
V. Az így kapott együtthatókat (jelöljük g-vel) variálni kezdjük és addig végezzük a variálást, amíg az így asszociált polinom nem osztja f-et, vagy $|g(1)| > |f(1)|$, vagy $deg(g) > deg(f)$.
VI. Ha sikerült a felbontás, akkor kiírjuk a *g* együtthatóit és megállunk, ha nem, akkor vesszük a következő osztót és visszatérünk (IV)-re.
VII. Ha az összes osztót kimerítettük és mégsem sikerült a felbontás, akkor növeljük az alapot (*B*-t) egy kvantummal és visszatérünk a (II)-re.
VIII. Háromszori alapnövelés után kiírjuk, hogy *f* irreducibilis és megállunk.

De a tesztelés során mindig sikerült felbontanom az összetett polinomokat.

Tehát az a következtetés vonható le, hogy vagy ki kell próbálni az összes osztó felét, vagy a prímosztók kettőnkénti kombinációival próbálkozni más-más alapban. A teljes algoritmus beprogramozásához a következő eljárások szükségesek: prímtényezőre bontás, két egész együtthatós polinom maradékos osztása, egy szám összes osztóinak előállítása. Az első kettő (prímtényezőre bontás és két egész együtthatós polinom osztása, jól ismert a „programozási folklórban" – Knuth könyvében is vannak hatékony eljárások). Egy szám összes osztóinak előállítására viszont mi találtuk ki az itt közölt rekurzív megoldást.

A programban a *kesz* eljárás végzi a variálásokat és az *osztaselle* a polinomok oszthatóságát ellenőrzi.

A felbontásra kijelölt polinom bevitele egy $deg(f)+2$ elemű vektorba történik, ahol $deg(f)$, *f* polinom foka és az együtthatókat fordított sorrendbe írjuk be. Például *f*-el jelölve a vektort, *f[1]* a polinom fokszámát tartalmazza, *f[2]* a szabad tagot, *f[3]* az *X*-es tag együtthatóját, ... *f[deg[f]+2]* a domináns tagot, vagyis a legmagasabb fokszámú *X*-es tag együtthatóját. A kiírás is így történik, csak nem írattuk ki a fokszámokat.

Ezen eljárásokat lehetne javítani, optimalizálni, mi ezt nem tettük még meg, mert az eredeti algoritmusra összpontosítottunk. Ugyanakkor, a program nincs lekezelve a túlcsordulás végett sem. A http://www.csik.sapientia.ro/ghkar/oktatok/olahgalrobert.html helyről letölthető program csak n < 20-ra működik, ahol n a felbontásra váró polinom fokszáma. Még az is megtörténhet, hogy az olvasó a tesztelés során egy olyan polinomot kap, amelyik reducibilis, a mi programunk mégsem tudja felbontani (mi 50 polinomra leteszteltük a programot jó eredménnyel). Ebben az esetben a hibát a programunkban keressék és ne az algoritmusban.

A mellékelt pszeudokódban csak a lényegesebb eljárásokat adtuk meg. Használtunk egy *Power* nevű hatványfüggvényt k^m kiszámítására, továbbá a *polioszt* nevű eljárás két egész együtthatós polinom maradékos osztására szolgál. Ezeket nem részletezzük.

A *primt* nevű eljárás egy egész szám prímtényezőrebontása és egy Vandermonde-típusú mártix feltöltésére szolgál. Ha $n = p_1^{\alpha 1} p_2^{\alpha 2} \ldots p_n^{\alpha n}$ akkor az összes osztók száma egyenlő: $(\alpha_1+1)(\alpha_2+1)\ldots(\alpha_n+1)$.

p_1	p_2	...	p_n
p_1^2	p_2^2	...	p_n^2
...	...	...	...
$p_1^{\alpha 1}$	$p_2^{\alpha 2}$	...	$p_n^{\alpha n}$

Az eljárásunkban *dim*-mel jelöltük az alfákat, és a prímtényezőket egy Vandermonde-típusú mátrixba raktároztuk a hatványaik sorrendjében.

KÖNYVÉSZET

[1.] Knuth D. E.: *A számítógép-programozás művészete*, I, II, III kötet, Műszaki Kiadó, Budapest, 1992.

Egyensúlyi zónák kaotikusságának vizsgálata a korlátozott háromtest problémában, konzervatív integrátorral megvalósított, FLI és SALI felületekkel

Survey of Chaoticity of Zones Near Equilibrium Points for the Restricted Problem of Three Bodies, with *SALI* and *FLI* Surfaces, Calculated with a Conservative Integrator

Examinarea comportării haotice a zonelor limitrofe punctelor de echilibru pentru problema restrânsă a celor trei corpuri, cu suprafețe *FLI* şi *SALI*, calculate cu un integrator conservativ

Drd. KOVÁCS Barna

Al. Papiu Ilarian Nemzeti Kollégium, Bolyai Farkas Elméleti Líceum, Marosvásárhely, Románia
t_barna_ro@yahoo.com

ABSTRACT

Chaos detection methods based on calculation of Fast Lyapunov Indicator (FLI) and Small Alignment Index (SALI) are known as fast ones. The celerity of these methods can be improved by the use of a fast and precise numerical integration method. With an integrator based on the conservative integration algorithm, we constructed FLI and SALI surfaces near equilibrium points of the restricted three body problem, and examined the chaotic nature of these zones.

Kulcsszavak: numerikus integrálás, korlátozott háromtest-probléma, káosz, SALI, FLI

1. A KONZERVATÍV INTEGRÁLÁS ELEMEI

A Naprendszert leíró egyenletek egy konzervativ dinamikai rendszert alkotnak mivel, súrlódás hiányában, tulajdonképpen elhanyagolható súrlódás mellett, a rendszer teljes energiája és impulzusnyomatéka megmarad. Egy numerikus integrátor hatásfoka lemérhető, ahogy a rendszer összenergiájat az integrálás során konzerválja. A konzervatív integrálás egy ilyen numerikus módszer, melyet először *B.A. Shadwick*, *W.F. Buell* és *J.C. Bowman (1999)* mutatott be. A módszer alapötlete a következő: a rendszer energiáját leíró egyenlet változóit egy olyan térbe kell transzformálni, amelyben az energiát leíró egyenlet lineárisan függ a transzformált változóktól. A transzformációt egy, az új változókra alkalmazott, integrációs lépés követi valamely konvencionális integrátorral, majd a kapott új értékeket visszatranszformáljuk az eredeti rendszerbe. Az így kapott értékek adják a konzervatív módszer következő lépésének kezdőértékeit. Legyen az $\mathbf{x}' = \mathbf{f}(\mathbf{x}, t)$ differenciálegyenlet–rendszer. A *konzervatív prediktor–korrektor algoritmus* során egy olyan $\xi = \mathbf{T}(\mathbf{x})$ transzformációt alkalmazunk, amely által a konzerválódó mennyiség a ξ_i, i=1,. . . ,n változóktól lineárisan függ. Az eredeti térben alkalmazzuk az

$$\tilde{\mathbf{x}} = \mathbf{x}_0 + \tau \mathbf{f}(\mathbf{x}_0, t)$$

prediktort, míg a transzformált térben a

$$\xi(t+\tau) = \xi_0 + \frac{\tau}{2}\left(\mathbf{T}'(\mathbf{x}_0)\mathbf{f}(\mathbf{x}_0, t) + \mathbf{T}'(\tilde{\mathbf{x}})\mathbf{f}(\tilde{\mathbf{x}}, t+\tau)\right)$$

korrektort, ahol $\xi_0 = \mathbf{T}(\mathbf{x}_0)$, $\mathbf{T}'$ pedig a $\mathbf{T}$ transzformációs függvény deriváltja. Az új $\mathbf{x}$ vektort az

$$\mathbf{x}(t+\tau) = \mathbf{T}^{-1}(\xi(t+\tau))$$

fordított transzformációból kapjuk. Megjegyezzük, hogy az itt leírt és alkalmazott módszer konzerválja a rendszer invariánsait, de nem konzerválja szükségszerűen a fázistér térfogatát: a *konzervatív integrálási módszer nem szimplektikus módszer*!

2. A KORLÁTOZOTT HÁROMTEST PROBLÉMA (KHTP) – MOZGÁSEGYENLETEK

A KHTP a következőképpen fogalmazható meg: *Adottak P_1,P_2 és P_3, tömegpontok, melyek között csak a Newton–féle gravitációs vonzóerők hatnak. Feltételezzük, hogy a P_3 tömege olyan kicsi, hogy hatása a P_1-re és P_2-re elhanyagolható, illetve azt, hogy P_1 és P_2 a közös súlypont körül egyenletes körmozgást végez. P_3 mindig ezen körmozgás síkjában található. Meghatározandó a P_3 mozgása.*

A mozgásegyenletek:

$$\begin{cases} \ddot{x} - 2\dot{y} = \dfrac{\partial \Omega}{\partial x} \\ \ddot{y} + 2\dot{x} = \dfrac{\partial \Omega}{\partial y} \end{cases}, \quad \text{ahol :} \tag{1}$$

$$\Omega = \frac{1}{2}\left[(1-\mu)r_1^2 + \mu r_2^2\right] + \frac{1-\mu}{r_1} + \frac{\mu}{r_2},$$

$$\mu = \frac{m_1}{m_1 + m_2}$$

$$r_1 = \sqrt{(x-\mu)^2 + y^2}$$

$$r_2 = \sqrt{(x+1-\mu)^2 + y^2}$$

Az (1) rendszernek egy elsőintegrálja van, a *Jacobi integrál*: $C = 2\Omega - \left(\dot{x}^2 + \dot{y}^2\right)$, ahol C a *Jacobi-konstans*. Egy adott pályára a *Jacobi-konstans* értéke állandó, így a numerikus integrálás során ennek a számított értéke, illetve értékváltozása, esetleg értékmegmaradása az integrálás pontosságát adja meg.

A *KHTP* Hamilton függvénye:

$$H = \frac{1}{2}\left(\dot{x}^2 + \dot{y}^2\right) - \frac{1}{2}\left(x^2 + y^2\right) - \frac{1-\mu}{r_1} - \frac{\mu}{r_2}, \tag{2}$$

Ha a $q_1 = x, \quad q_2 = y, \quad p_1 = \dot{x} - y, \quad p_2 = \dot{y} + x$ helyettesítéseket alkalmazzuk, akkor a Hamilton-függvény kanonikus alakját kapjuk:

$$H = \frac{1}{2}\left(p_1^2 + p_2^2\right) + p_1 q_2 - p_2 q_1 - \frac{1-\mu}{r_1} - \frac{\mu}{r_2}. \tag{3}$$

A mozgásegyenletek:

$$\begin{cases} \dot{q}_1 = \dfrac{\partial H}{\partial p_1} \\ \dot{q}_2 = \dfrac{\partial H}{\partial p_2} \\ \dot{p}_1 = -\dfrac{\partial H}{\partial q_1} \\ \dot{p}_2 = -\dfrac{\partial H}{\partial q_2} \end{cases}, \qquad \begin{cases} \dot{q}_1 = p_1 + q_2 \\ \dot{q}_2 = p_2 - q_1 \\ \dot{p}_1 = p_2 - \dfrac{1-\mu}{r_1^3}(q_1 - \mu) - \dfrac{\mu}{r_2^3}(q_1 + 1 - \mu) \\ \dot{p}_2 = -p_1 - \dfrac{1-\mu}{r_1^3} q_2 - \dfrac{\mu}{r_2^3} q_2 \end{cases} \tag{4}$$

A (2), (3) és 4 egyenletekből kapjuk, hogy:

$$H = \frac{1}{2}\left(\dot{q}_1^2 + \dot{q}_2^2\right) - \frac{1}{2}\left(q_1^2 + q_2^2\right) - \frac{1-\mu}{r_1} - \frac{\mu}{r_2} \tag{5}$$

A *KHTP* integrálása a konzervatív integrátorral

A rendszer (4) *prediktora*:

$$\begin{cases} \tilde{q}_i = q_i + \dot{q}_i \tau \\ \tilde{p}_i = p_i + \dot{p}_i \tau \end{cases} \qquad \text{i=1,2} \tag{6}$$

Legyen:

$$\xi_1 = \frac{1}{2} q_1^2, \quad \xi_2 = \frac{1}{2} q_2^2, \quad \xi_3 = \frac{1}{2} \dot{q}_1^2 - \frac{1-\mu}{r_1} - \frac{\mu}{r_2}, \quad \xi_4 = \frac{1}{2} \dot{q}_2^2, \tag{7}$$

A Hamilton – függvény, (5) és (7) figyelembevételével, a:

$$H = -\xi_1 - \xi_2 + \xi_3 + \xi_4 \tag{8}$$

kifejezésre transzformálódik

A (4) rendszer *korrektora*:

$$\xi_i \left(t + \tau \right) = \xi_i + \frac{\tau}{2} \left(\dot{\xi}_i + \dot{\tilde{\xi}}_i \right) \quad \text{i= 1. . .4} \tag{9}$$

Az egyes új ξ_i értékek kiszámítása után az új helyzet- és sebességértékeket a (10) transzformációkkal számoljuk ki

$$\begin{aligned} q_1 &= \text{signum}(\tilde{q}_1) \sqrt{2\xi_1} \\ q_2 &= \text{signum}(\tilde{q}_2) \sqrt{2\xi_2} \\ p_1 &= -q_2 + \text{signum}(\tilde{p}_1 + \tilde{q}_2) \sqrt{2\xi_3 + \frac{2(1-\mu)}{r_1} + \frac{2\mu}{r_2}} \\ p_2 &= q_1 + \text{signum}(\tilde{p}_2 - \tilde{q}_1) \sqrt{2\xi_4} \end{aligned} \tag{10}$$

3. A GYORS LYAPUNOV MUTATÓ (*FLI* – FAST LYAPUNOV INDICATOR) – A KÁOSZ ÉSZLELÉSÉNEK ESZKÖZE

Legyen :

$$\dot{\mathbf{x}} = F(\mathbf{x}(t)), \quad \mathbf{x} \in \Re^n, \quad t \in \Re,$$

$$\begin{cases} \dot{\mathbf{x}} = F(\mathbf{x}(t)) \\ \dot{\mathbf{v}} = \dfrac{\partial \mathbf{F}}{\partial \mathbf{x}} (\mathbf{x}(t)) \mathbf{v} \end{cases}$$

A kaotikus, illetve gyengén kaotikus és periodikus pályák azonosítására *Froeschlé és szerzőtársai* (1997) a $\log \|\mathbf{v}(t)\|$ értéket használták mint *gyors Lyapuov mutatót (Fast Lyapunov Indicator – FLI)*, egy adott T pillanatban. *Froeschlé és Lega* (2000) a *FLI* egy újabb definícióját adtak meg, a $\log \|\mathbf{v}(t)\|$ értékek átlagát számolva a $[T - \Delta t, T]$ intervallumban, míg *Froesché és Lega* (2001) a gyors *Lyapunov* mutató egy harmadik definícióját adtak meg: $\text{FLI}(\mathbf{x}(0), \mathbf{v}(0), T) = \sup_{0<k<T} \log \|\mathbf{v}(k)\|$.

Fouchard és szerzőtársai (2002) által végzett számítások kimutatták, hogy egy kaotikus pálya esetében a *FLI* értékének növekedése exponenciális, míg a reguláris, nem rezonáns pályák esetében a *FLI* időbeni növekedése lineáris. Periodikus pályák esetében a *FLI* éréke egy átmeneti változás után egy konstans értéket vesz fel, egy, a grafikonon az *Ox* tengellyel közel párhuzamos fennsíkot (*plateau*) képezve. Ugyanitt a szerzők megállapítják, hogy nincs egy adott *FLI* érték, amely egyértelműen elválasztja a gyengén kaotikus pályákat a reguláris és rezonáns pályáktól.

4. A *SALI* (SMALLER ALLIGNMENT INDEX) – A KÁOSZ ÉSZLELÉSÉNEK ESZKÖZE

A *SALI* módszert *Skokos Ch*. (2001) vezette be. A *SALI* segítségével sikeresen állapították meg egyes Hamilton-rendszerekhez tartozó mozgások kaotikusságát. A módszert sikeresen alkalmazták többek között az égi mechanikában, pályák kaotikusságának kimutatására.

A *SALI* definiálásához tekintsük egy konzervatív dinamikus rendszer k dimenziós fázisterét. A rendszer szabadságfoka legyen N, így $k=2N$. Az $\mathbf{X}(0) = \left(x_1(0),\ x_2(0),\ ...x_k(t)\right)$ kezdeti feltételekkel rendelkező pályát a Hamilton féle mozgásegyenletek határozzák meg, ezeknek az általános alakja:

$$\frac{d\mathbf{X}(t)}{dt} = F(\mathbf{X}(t)) \quad . \tag{11}$$

ahol $\mathbf{X}(t) = \left(x_1(t),\ x_2(t),\ ...x_k(t)\right)$-vel jelöltük a pálya helyét a fázistérben a t időpillanatban.

A kaotikusság vizsgálata szükségessé teszi a $\mathbf{V}(t) = \left(dx_1(t),\ \mathrm{d}x_2(t),\ ...\mathrm{d}x_k(t)\right)$ kitérés-vektor vizsgálatát. Esetünkben a kitérés-vektor vizsgálata a

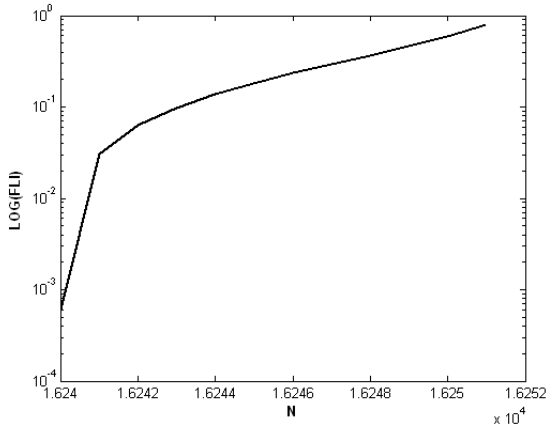
$$\frac{d\mathbf{V}(t)}{dt} = DF(\mathbf{X}(t)) \cdot \mathbf{V}(t) \tag{12}$$

variációs egyenletekkel történik. Itt DF jelöli a (4) rendszer mellé rendelt, a pálya pontjaira számított *Jacobi*-matrixot.

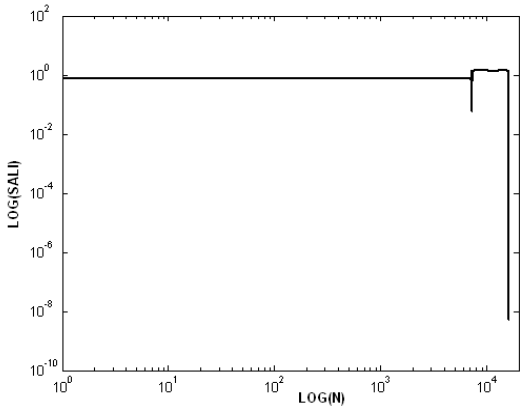
A *SALI* kiszámításához tekintsünk két, $\mathbf{V}_1(0)$ és $\mathbf{V}_2(0)$ kitérés-vektort. A pálya változása során minden integrációs lépésre kiszámítjuk a

$$SALI(t) = \min\left\{\left\|\frac{\mathbf{V}_1(t)}{\|\mathbf{V}_1(t)\|} + \frac{\mathbf{V}_2(t)}{\|\mathbf{V}_2(t)\|}\right\|, \left\|\frac{\mathbf{V}_1(t)}{\|\mathbf{V}_1(t)\|} - \frac{\mathbf{V}_2(t)}{\|\mathbf{V}_2(t)\|}\right\|\right\} \tag{13}$$

értéket. Ha a vizsgált pálya kaotikus, akkor a *SALI* értéke exponenciálisan tart a nullához, ellenkező esetben, vagyis periodikus illetve kváziperiodikus pályák esetében, a (13)-mal megadott számolt érték fluktuációs mozgást végez egy pozitív, nem nulla középérték körül.



1. ábra. Kaotikus pálya *log(FLI)* változása a lépésszám (*N*) függvényében. Infinitezimális test - Nap – Jupiter rendszer, *X_0 = -0.994430522142340, Y_0 = 0.00720*, a *Jacobi*-konstans értéke *C=3.038441716255740010*



2. ábra. Kaotikus pálya *log(SALI)* változása a lépésszám *log(N)* függvényében Infinitezimális test - Nap – Jupiter rendszer, *X_0 = -0.994430522142340, Y_0 = 0.00720*, a *Jacobi*-konstans értéke *C=3.038441716255740010*

5. LAGRANGE-FÉLE EGYENSÚLYI PONTOK

A *KHTP*-nak a $\dot{x} = 0,\ \dot{y} = 0,\ \ddot{x}=0,\ \ddot{y}=0$ kezdeti feltételekre kapott megoldásait egyensúlyi megoldásoknak nevezzük. Ha $y \neq 0$, akkor az eredmény az L_4 és L_5 librációs pontok koordinátái: $L_4 = L_4(\mu - \frac{1}{2}, \frac{\sqrt{3}}{2})$ illetve $L_5 = L5(\mu - \frac{1}{2}, -\frac{\sqrt{3}}{2})$. Az L_4 és L_5 librációs pontok a P_1 illetve P_2 pontokkal egy-egy egyenlőszárú háromszöget alkotnak. Ha $y = 0$, akkor az L_1, L_2 és L_3 pontokat kapjuk. Az L_1 pont x koordinátáját az

$$\begin{aligned} &x = \mu - 1 - \theta, \\ &\theta^5 + (3-\mu)\theta^4 + (3-2\mu)\theta^3 - \mu\theta^2 - 2\mu\theta - \mu = 0 \end{aligned} \tag{14}$$

rendszerből kapjuk, az egyenletet megoldva.

Hasonlóan az L_2 pont x koordinátáját az

$$\begin{aligned} &x = \mu - 1 + \theta, \\ &\theta^5 - (3-\mu)\theta^4 + (3-2\mu)\theta^3 - \mu\theta^2 + 2\mu\theta - \mu = 0 \end{aligned} \tag{15}$$

rendszerből kapjuk, végül az L_3 pont x koordinátája az

$$\begin{aligned} &x = \mu + \theta, \\ &\theta^5 + (2+\mu)\theta^4 + (1+2\mu)\theta^3 - (1-\mu)\theta^2 - 2(1-\mu)\theta - \mu = 0 \end{aligned} \tag{16}$$

egyenlet megoldása.

6. FLI ÉS SALI FELÜLETEK, MINT A PERIODIKUS PÁLYÁK ÉSZLELÉSÉNEK ESZKÖZEI

Egy adott L_i ($i=1..4$) körüli felület pontjait egy 100×100-as pontrácsba foglaltuk, amelynek minden pontjából egy-egy infinitezimális testet indítottunk, az adott L_i-nek megfelelő, állandó *Jacobi*-konstans mellett. A konzervatív integrátorral minden pálya esetében meghatároztuk a *FLI* és *SALI* változását. Az integrálás a pontrács minden pontja esetében, amennyiben az reguláris volt, 100 időegységet tartott ($t=100$). Ha a pálya időközben kaotikusnak bizonyult, vagyis a *SALI* értéke 10^{-8} nagyságrendűre csökkent, akkor az integrálást megállítottuk. Az eredményeket egy-egy 100×100-as mátrixban tároltuk, melyeket aztán felületként ábrázoltunk, a *Matlab 6.0* szoftver segítségével.

A kapott felületeket vizsgálva megállapítottuk, hogy viszonylag nagy kiterjedésű, összefüggő, stabil felületekből, amelyek egy viszonylag kis *FLI* értékkel, $FLI \in [-4, -1]$, illetve egy viszonylag nagy SALI értékkel, $SALI \in [1, 4]$, jellemezhetők, jó eséllyel indíthatók periódikus vagy kváziperiodikus pályák.

Megjegyzendő, hogy a felületek előállítása során, a *Jacobi*-konstans pontossága a pontrács minden pontja esetében 10^{-15} nagyságrendű volt.

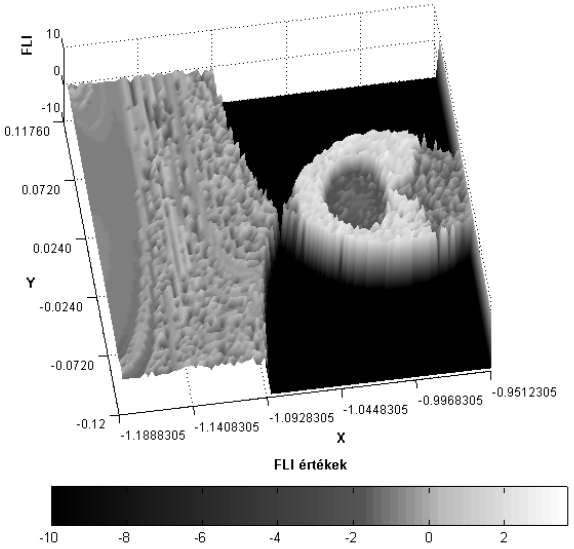
A periodikus/kváziperiodikus pályákkal kecsegtető pontokat úgy azonosítottuk be, hogy például a *FLI*–felületen megkerestük a kis *FLI*–értékű, „sima” és összefüggő felületrészeket. A továbbiakban ezekből a kis *FLI*–értékű pontokból indított infinitezimális test pályájának regularitását vizsgáltuk. A *SALI* felülettel hasonlóan jártunk el, nagy értékeket keresve a felületen.

Egy reguláris, periodikus vagy kváziperiodikus pálya *FLI* illetve *SALI* értékeinek változásai az *5.* és *6.* ábrákon láthatók. A kezdőpont kiválasztása a *FLI*-felület fent ismertetett kiértékelésével történt. Az integrálás lépésszáma (5. és 6. ábra) $N \approx 2.25 \cdot 10^9$, ami közel 2.25 millió földi évnek felel meg. Az integrálás során a *Jacobi*-konstas hibája 10^{-15} nagyságrendű volt.

7. KÖVETKEZTETÉSEK

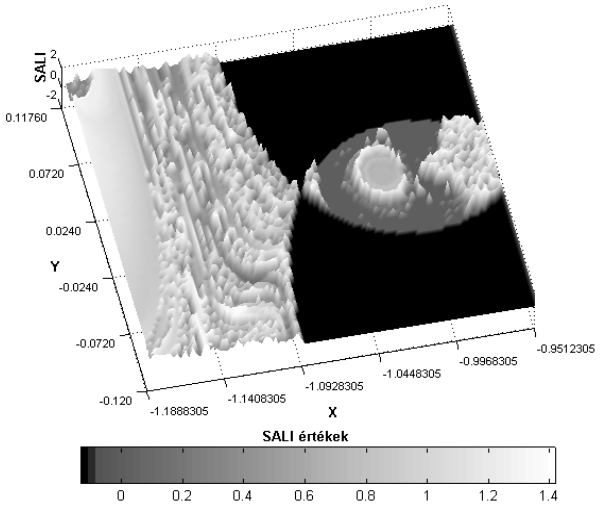
A konzervatív integrátor gyors és hatékony eszköz konzervatív dinamikus rendszerek numerikus integrálására. A konzervatív integrátorral előállított és ismertetett *FLI*, illetve *SALI* felületek egy összképet adnak az egyes egyensúlyi felületek kaotikusságáról. A felületeket vizsgálva, sikeresen azonosítottunk olyan

pontokat, amelyekből kváziperiodikus pályára állíthattuk az infinitezimális testet. Megállapítható, hogy a konzervatív integrátorral előállított *FLI*, és *SALI* felületek alkalmasak reguláris pályák észlelésére.



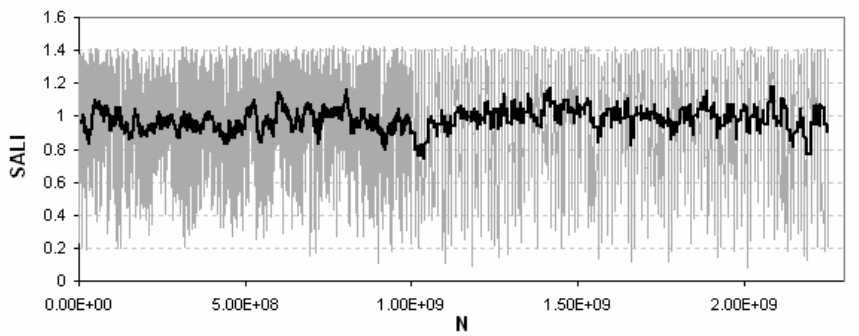
3. ábra. *FLI* felület az L_1 pont környezetében. Infinitezimális test–Nap–Jupiter rendszer.

C = 3.038441716256784280

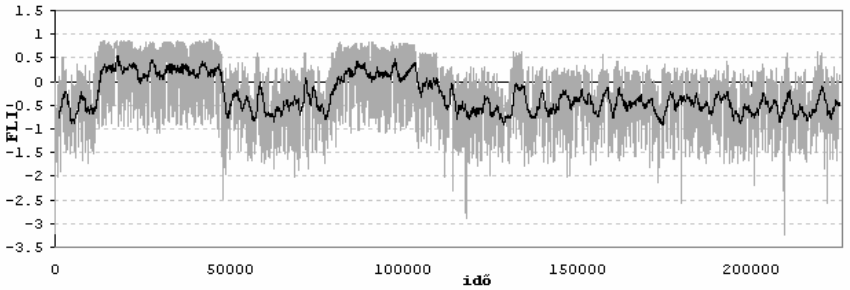


4. ábra. *SALI* felület az L_1 pont környezetében. Infinitezimális test–Nap–Jupiter rendszer.

C = 3.038441716256784280



5. ábra. Reguláris pálya *SALI* változása a lépésszám függvényében. Infinitezimális test–Nap–Jupiter rendszer, X_0 = *-1.064030522142340,* Y_0 = *0.00240*, a *Jacobi*-konstans értéke *C=3.038441716255740010*

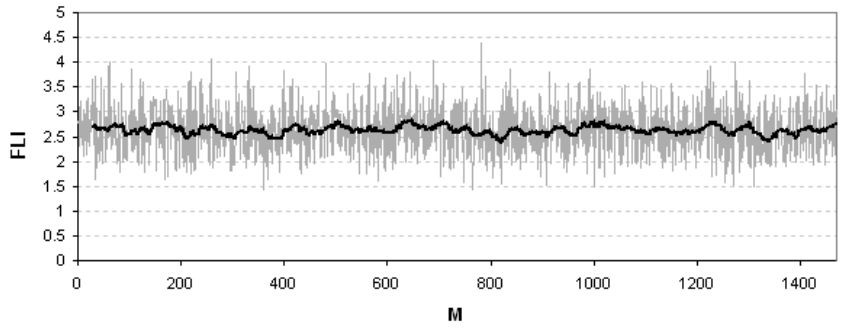


6. ábra. Reguláris pálya *FLI* változása a lépésszám függvényében. Infinitezimális test–Nap–Jupiter rendszer, $X_0 = -1.064030522142340$, $Y_0 = 0.00240$, a *Jacobi*-konstans értéke $C=3.038441716255740010$

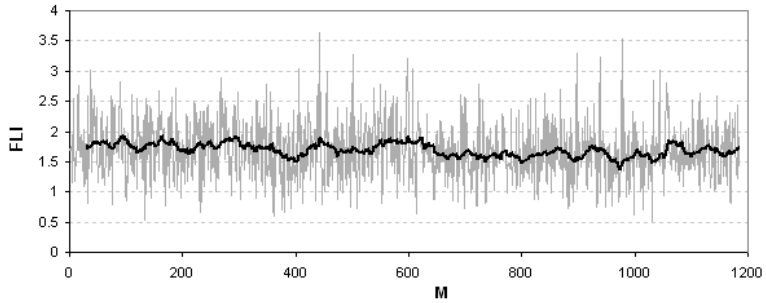
Felvetődik a kérdés: milyen *FLI* értékek mellett tekinthetjük a pályát kaotikusnak. A válasz érdekében feljegyeztük a *FLI* értékét abban a pillanatban, amikor a *SALI* mutató a felület készítésekor a káoszt kimutatta, azaz a 10^{-8} érték alá esett. Az eredmények összesítését három rendszerre Föld–Hold, Nap–Jupiter illetve 51 Peg–51 Peg b környezetében mozgó infinitezimális test estében végeztük el. Az eredményeket, a káosz megjelenésekor lejegyzett kritikus *FLI* értékeket a 7., 8. és 9. ábrákon mutatjuk be. Az egyes rendszerek *FLI* értékeinek eloszlását a 10., 11. és 12. ábrák szemléltetik. Megfigyelhető, hogy mindhárom rendszer esetében más és más a kritikus *FLI* érték átlaga. A kritikus *FLI* érték és az egyes rendszerek $\mu = \dfrac{m_1}{m_1 + m_2}$ tömegaránya közötti esetleges összefüggés megállapítására tekintsük az 1. táblázatot, illetve a 7., 8. és 9. ábrákat:

	Föld–Hold–kis test	Nap–Jupiter–kis test	PEG 51–PEG 51 b–kis test
μ	*0.012290969899665600*	*0.000953875*	*0.00042596708015267*
$FLI_{átlag}$	*2.63198845*	*1.693380413*	*1.365341897*
$FLI_{maximum}$	*6.388382706*	*3.63089685*	*3.166028883*
$FLI_{minimum}$	*1.430667576*	*0.550352528*	*0.20429205*

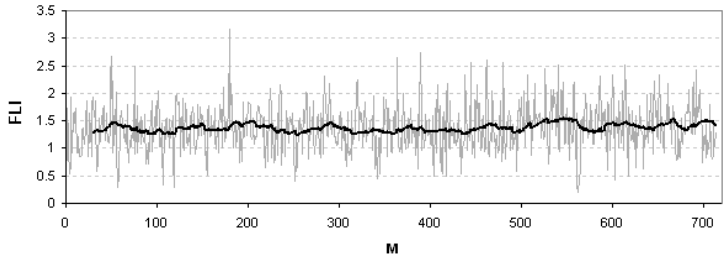
1. táblázat



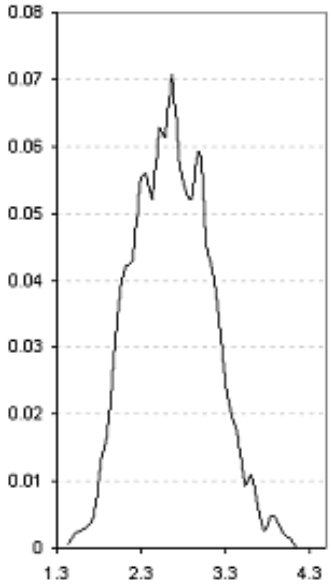
7. ábra. *FLI* értékek a *SALI* által kimutatott káosz megjelenésekor. Föld–Hold–kis test rendszer. $FLI_{átlag}=2.63198845$, legkisseb érték $FLI_{min}=1.430667576$, legnagyobb érték $FLI_{max}=6.388382706$



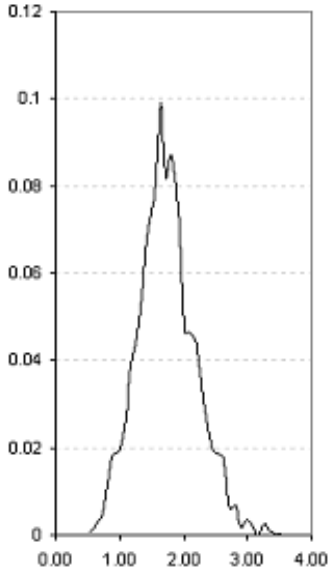
8. ábra. *FLI* értékek a *SALI* által kimutatott káosz megjelenésekor. Nap–Jupiter–kis test rendszer. $FLI_{átlag}= 1.693380413$, legkissebb érték $FLI_{min}= 0.550352528$, legnagyobb érték $FLI_{max}= 3.63089685$



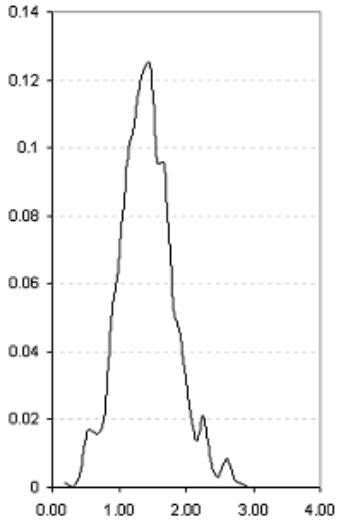
9. ábra. *FLI* értékek a *SALI* által kimutatott káosz megjelenésekor Peg 51–Peg 51b–kis test rendszer. $FLI_{átlag}$*= 1.365341897*, legkissebb érték FLI_{min}*= 0.20429205,* legnagyobb érték FLI_{max}*= 3.166028883*



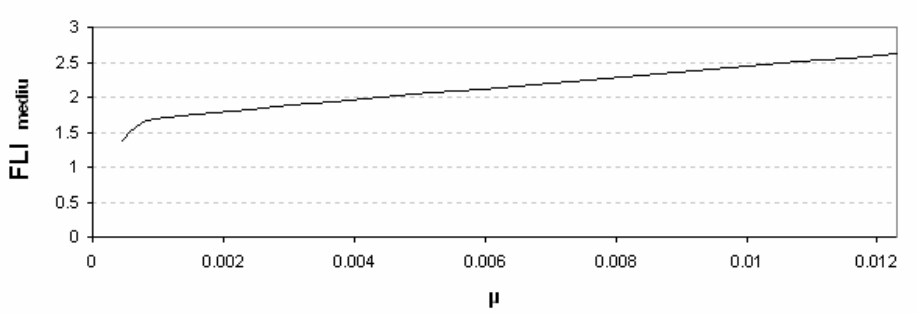
10. ábra. Kritikus *FLI* értékek spektruma, *Föld–Hold–kis test* rendszer



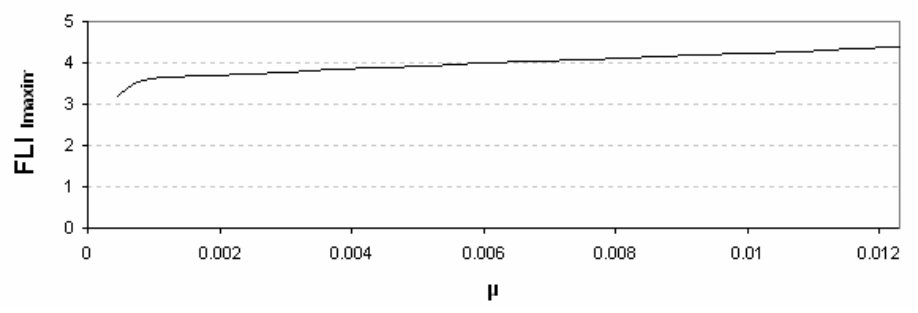
11. ábra. Kritikus *FLI* értékek spektruma, *Nap–Jupiter–kis test* rendszer



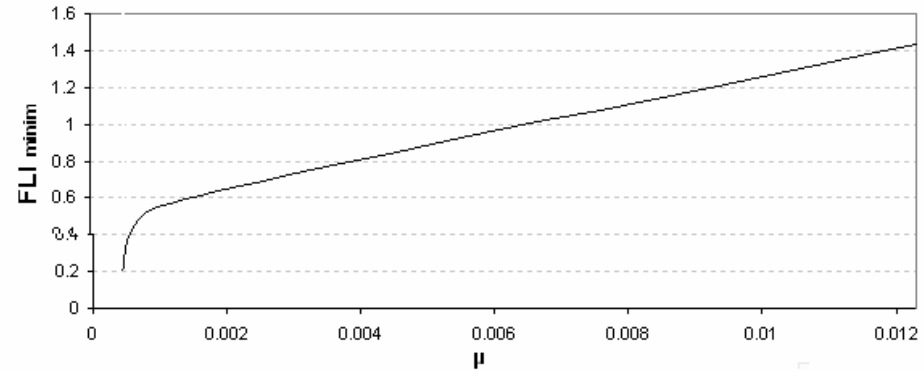
12. ábra. Kritikus *FLI* értékek spektruma, *PEG 51–PEG 51 b–kis test* rendszer



13. ábra. A kritikus *FLI* átlagának változása a μ tömegarány függvényében



14. ábra. A kritikus *FLI* maximumának változása a μ tömegarány függvényében



15. ábra. A kritikus *FLI* minimumának változása a μ tömegarány függvényében

A 13., 14. és 15. ábrákat figyelembe véve megállapítható, hogy *létezik egy lineáris függőség a tömegarány és a kritikus FLI érték között.*

KÖNYVÉSZET

[1.] Froeschlé, C., Lega, E. and Gonczi R.: *Fast Lyapunov indicators. Application to asteroidal motion*, In: *Celest. Mech. Dyn. Astron.* **67**: 41–62, 1997.

[2.] Froeschlé, C., and Lega, E.*: On the structure of symplectic mappings. The fast Lyapunov indicator : a very sensitive tool*, In: *Celest. Mech. Dyn. Astron.* **78**: 167–195, 2000.

[3.] Froeschlé, C., and Lega, E.: *On the relationship between fast Lyapunov indicator and periodic orbits for symplectic mappings*, In: *Celest. Mech. Dyn. Astron.* **81**: 129–147, 2001.

[4.] Fouchard, Marc; Lega, Elena; Froeschlé, Christiane; Froeschlé, Claude: *On the Relationship Between Fast Lyapunov Indicator and Periodic Orbits for Continuous Flows*, In: *Celest. Mech. Dyn. Astron.* **83**: 205–222, 2002.

[5.] Érdi B.: *Numerikus megoldások a korlátozott háromtest problémában*, In: *Androméda* **3**: 3–5, 2002.

[6.] B.A. Shadwick. J.C. Bowman, P.J. Morrison: *Exactly conservative integrators*, In: *SIAM J. Appl. Math.* **59**: 1112–1133, 1999.

[7.] Kotovych O. Bowman J.: *An Exactly Conservative Integrator for the n-Body Problem*, In: *J. Phys. A: Math. Gen.* **35**: 7849–7863, 2002.

[8.] Skokos Ch.: *Alignment indices: A new, simple method for determining the ordered or chaotic nature of orbits*, In: *J. Phys. A*, **34**: 10029–10043, 2001.

[9.] Szebehely V.: *The Theory of Orbits*, Academic Press, New York, London, 1967.

Számonkérési formák programozási versenyeken

Examining Methods of Programming Competitions

Metode de examinare la concursuri de programare

[1]Dr. TARCSI Ádám, [2]Dr. ZSAKÓ László

ELTE Budapest, Informatikai Kar
[1]ade@elte.hu, [2]zsako@ludens.elte.hu

ABSTRACT

In Hungary (and in other countries) the computer science competitions have had almost 30-year tradition. The majority of these are programming competitions in which relatively narrow amount of people participate. The remarkable part of informatics is about the applications, which requires a totally different knowledge. This paper is going to show the differences of the programming competitions. The application competitions are the topics of another presentation.

We think that the "without-computer" and "with-computer" done competitions play a very important role. Both types of the competitions have parts that expect a well defined knowledge from the students. At the national competitions it's practical to measure this knowledge at the first and partly at the second round. For the most talented students these competitions are preparations for the future of profession on Informatics thus they have to be contacted with similar exercises to the Olympic competitions. We mind that improving the informatics knowledge has to be gradually, as we will show it in our examples.

Kulcszavak: Informatika versenyek, számonlkérés, feladatok

1. INFORMATIKA VERSENYEK RÖVID TÖRTÉNETE MAGYARORSZÁGON

Magyarországon egy 1983-as próbálkozás után 1985-ben kezdődött az országos informatika versenyek sorozata [1]. Ugyanekkor indult például Bulgáriában [2] és Szlovákiában [3] is. Németországban ennél is régebben kezdődtek az informatikai versenyek [4].

Sok éven keresztül programozási versenyről beszélhettünk (Nemes Tihamér OKSzTV), három korcsoportban: 5-8., 9-10., illetve 11-12. osztályos tanulóknak. Az országos versenyek mindig 3 fordulósak: iskolai, regionális és országos fordulóval.

Az 1989-ben kezdődött International Olympiad in Informatics [5] és az 1994-ben indult Central-European Olympiad in Informatics* [6] miatt 1989-től a programozási versenyrendszer egy negyedik fordulóval, az olimpiai válogatóversennyel bővült.

A résztvevők számának alakulását mutatja a mellékelt 1. ábra.

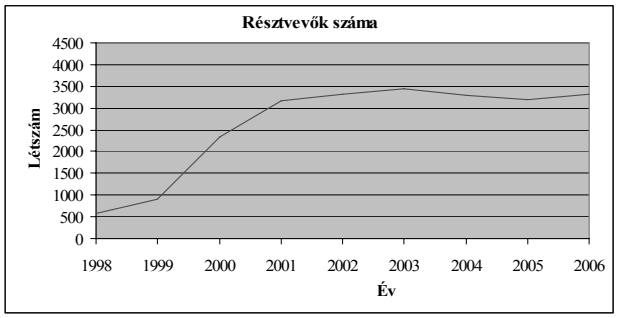
1. ábra

* Körülbelül ebben az időben indult több más regionális verseny, pl. Balkan Olympiad in Informatics, illetve Baltic Olympiad in Informatics.

A diagram érdekes jellemzője (de most nem témánk), hogy az induló versenyzők létszáma erős korrelációban áll a közoktatásban ajánlott, vagy kötelező informatika tanórák számával.

A Logo programozási nyelv széleskörű elterjedése miatt (bekerült a Nemzeti Alaptantervbe, illetve a Comenius Logo több száz iskolába eljutott) 1998-ban indult a Logo programozási verseny. Ez az évek során 4 korcsoportra tagozódott: 3-4., 5-6., 7-8., illetve 9-10. osztályos tanulóknak (a Nemes Tihamér versenyhez hasonlóan iskolai, regionális és országos fordulóval).

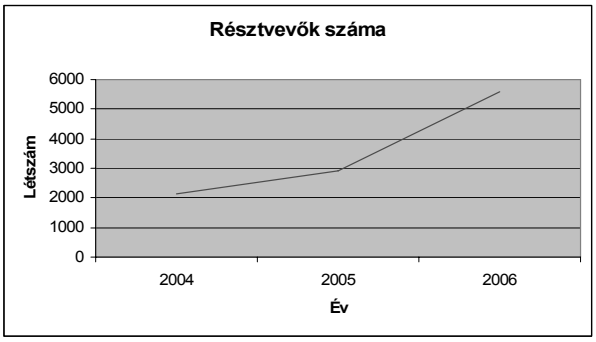
A mellékelt diagramon (2. ábra) látszik a gyors felfutás és a stabil 3500 körüli résztvevő szám.



2. ábra

A programozási versenyek sikere után több évvel jelentek meg másfajta informatikai versenyek. Az 1995-ben indult budapesti alkalmazói verseny (rajzolás, szövegszerkesztés, táblázatkezelés, adatbázis-kezelés, honlap-szerkesztés, prezentáció) 2004-től országos tanulmányi verseny, 2 korcsoportban 9-10., illetve 11-12. osztályos tanulók számára.

A mellékelt diagramon látszik, hogy a résztvevők száma szinte pillanatok alatt túlszárnyalta a másik két verseny indulóinak létszámát.



3. ábra

2. PROGRAMOZÁS ISMERETEK SZÁMONKÉRÉSI MÓDSZEREI (A NEMES TIHAMÉR OKSZTV FELADAT-TÍPUSAIN ILLUSZTRÁLVA)

Számítógép nélküli számonkérések

Az iskolákban tartandó első fordulóban a tanulók **analizáló** képességét tesszük próbára számítógép használata nélkül: 6-10 kisebb feladatot (algoritmus- vagy programrészletet, működési vázlatot) adunk, és olyan kérdésekre várunk választ, mint pl. (1) mit csinál? (2) milyen hibák vannak benne? (3) milyen feltételek mellett működik? (4) mi hiányzik belőle? (5) mire használjuk a változókat? (6) megoldja-e a kitűzött feladatot? (7) mit tudunk a változók értékeiről? stb. Ebben a fordulóban számítógép nem használható.

Ez a feladattípus az egyes országok informatikai versenyein ritkaság. Jellemzően a 15-20 évvel ezelőtti versenyeken fordul elő (pl. Bulgária, Németország, Szlovákia [2,3,4]). A magyar programozási versenyeken azonban mind a mai napig megmaradt. Ennek elsődleges oka nem a hagyomány, hanem az informatika oktatás célja. Úgy gondoljuk, hogy a mindenkinek szóló informatikában is szükség van algoritmizálási ismeretekre. Itt azonban az elsődleges cél az algoritmusok megértési és végrehajtási képessége. Tömeges verseny esetén emiatt ezt a képességet kell mérni a verseny első fordulójában. Valószínűleg hasonló céllal vannak jelen a *Theoretical problems* a litván olimpiákon is [9]. Hasonló az Australian Informatics Competition: Sample Questions, itt azonban a válaszok is megjelennek, tesztszerűen, és a versenyzőknek csak választani kell közülük [11].

E különlegesség miatt itt kicsit részletesebben foglalkozunk ezzel a feladattípussal.

Egyik lehetőség egy algoritmus működésével kapcsolatos kérdések feltétele. Ez elképzelhető úgy is, hogy nem adjuk meg, hogy az algoritmusnak mi a feladata. Ilyenkor kérdezhetjük azt, hogy adott bemenetre milyen eredményt kapunk (pl. First Bulgarian National Olympiad, Problem 2 [7]), illetve azt, hogy adott kimenet esetén mi lehetett a bemenet (pl. First Bulgarian National Olympiad, Problem 4).

A mi példánkban megmondjuk mi a feladat, majd kétféle kérdést teszünk fel:

- mi a helyes működés feltétele, illetve ha a feltétel nem teljesül, akkor mit kapunk eredményként;
- helyes működés esetén mik a szélsőséges esetek a futási időre?[†]

Az alábbi algoritmus két rendezett vektor (A, B) elemeiből állít elő egy újabb rendezett vektort, amelyben azok az elemek szerepelnek, amelyek legalább az egyikben előfordulnak. Az eredmény minden elemének különbözőnek kell lennie.

```
Összefuttatás(A,N,B,M,C,K):
   I:=1; J:=1; K:=0
   while I≤N and J≤M do
      K:=K+1
      if A(I)<B(J) then C(K):=A(I); I:=I+1
         elif A(I)=B(J) then C(K):=A(I); I:=I+1; J:=J+1
         else C(K):=B(J); J:=J+1
      endif
   endwhile
endproc.
```

A. Milyen feltételeknek kell teljesülniük A-ra és B-re az eljárás helyes működéséhez?
B. Milyen hibát okoz az eredményben, ha ezek a feltételek nem teljesülnek?
C. Helyes működés esetén, rögzített N és M esetén milyen A és B vektorra a leggyorsabb, illetve a leglassúbb az algoritmus?

Egy másik feladattípus esetén az algoritmust szövegesen közöljük, majd a megértését próbáljuk ellenőrizni. A lehetséges kérdések itt az adatokra vonatkozó invariáns állítással, a bemenet és a kimenet közötti összefüggéssel, az algoritmus végrehajtása befejeződésének feltételével kapcsolatosak.

Egy dobozban fekete és fehér borsszemeket tárolunk. Véletlenszerűen kiveszünk 3 szemet. Ha mindhárom fekete, akkor nem teszünk semmit. Ha két fekete van köztük, akkor mind a hármat visszarakjuk. Ha két fehér van köztük, akkor a feketét visszarakjuk. Ha mindhárom fehér, akkor pedig egy fehéret rakunk vissza. Ha kettőnél több bors maradt, akkor a maradékra a fenti algoritmus újra kezdődik.
A: Hogyan változik a fekete, illetve a fehér borsszemek száma az algoritmus végrehajtása során?
B. A borsszemek számának milyen lényeges tulajdonsága nem változik meg az algoritmus végrehajtása során?
C. Milyen kiinduló állapot esetén kerülhetünk végtelen ciklusba az algoritmus végrehajtása során?
D. Mitől függ, hogy a végén hány bors marad és azok milyen színűek?

A harmadik példánkban egy adatstruktúra megértését, a vele dolgozó algoritmussal kapcsolatos kérdések megválaszolását várjuk el.

Egy vasútvonal két állomásán egy nap feljegyeztük az összes, a másik felé áthaladó vonat indulási, illetve érkezési idejét. Feltételezzük, hogy a vonatok a vizsgált szakaszon nem előzhetik meg egymást és egymással szemben soha sem haladhat két vonat. N adatpárt ismerünk, mindegyik egy állomás sorszámot és egy időpontot tartalmaz, idő szerinti sorrendben. Ezek alapján a program találja ki, hogy melyik adatpárban van indulási és melyikben érkezési idő.
A. Mit ír ki az alábbi algoritmus?
B. Mi a Queue szerepe az algoritmusban?
C. Mi a szerepe a (*)-gal jelölt sornak?

```
SorInicializálás
while not eof? do
 Olvas(állomás,time)
 if empty?(Queue) then x:=állomás endif (*)
 if állomás=x then Sorba(time)
                  else Sorból(t); write(time-t)
  endif
endwhile
```

D. Mi lesz a sor tartalma lépésenként a ciklusmag elején, ha az adatok a következők:
(1,1),(1,3),(2,4),(1,4),(1,5),(2,5),(2,6),(2,7),(2,8),(2,9),(1,13),(1,14)?

[†] A versenyekről vett példák szövegét dőlten szedve, keretezve közöljük.

A következő példa nyelvek szintaktikai szabályainak megértését, a nyelvet generáló automata működésének megértését, a rekurzió fogalmának, a rekurzív kiszámítás módjának használatát kéri számon.

A FURA nyelvben (a szokásos kettes számrendszer helyett) három jelet (A, B és C betű) használnak a szavak leírására. Mivel ez egy mesterséges nyelv, a szavakat szigorú szabályok betartásával hozzák létre.

- *A kiinduló szó mindig az ABB szó.*
- *Bármely A-val kezdődő szó A mögötti része megduplázható (Ax..y szóból Ax..yx..y alkotható).*
- *Szó végi B betű helyébe C írható.*
- *BC szó helyére BBCC írható.*
- *BC szó törölhető.*
- *BCB szó helyére BB írható.*

A. A FURA nyelv szavai-e az AC, ABBCC, ABCCC, ABBBBBB szavak? Amelyik igen, arra add meg, hogy milyen szabályok alkalmazásával kapható az ABB szóból!

B. Fogalmazd meg, milyen szabályoknak kell teljesülnie egy szóra, hogy a FURA nyelv szava legyen!

Utolsó példánk pedig a dinamikus programozás elvének megértéséről, annak kézi kivitelezéséről szól.

Egy karaktersorozat tükörszó, vagy palindrom, ha szimmetrikus, azaz ha balról jobbra és jobbról balra olvasva azonos.

Például 2 karakter beszúrásával az S=„Ab3bd" karaktersorozat palindrommá alakítható („dAb3bAd" vagy „Adb3bdA" is lehet belőle). Kettőnél kevesebb karakter beszúrásával azonban ebből a karaktersorozatból nem állítható elő palindrom.

*Jelöljük **M(i, j)**-vel minden **(i, j) 1≤i≤j≤N** indexpárra, hogy az **S[i..j]=S[i]...S[j]** szó legkevesebb hány betű-beszúrással tehető tükörszóvá!*

A. Add meg, hogy** S=**"FAKANÁL" esetén hogyan néz ki az** M **mátrix!

B. Adj képletet** M(i, j) **kiszámítására!

A Nemzetközi Informatikai Diákolimpián egyszer történt kísérlet ilyen feladat kitűzésére, 1995-ben, Hollandiában [8], azóta az olimpiákon ezt a feladattípust senki sem támogatja. A mi feladataink ennél sokkal egyszerűbbek, rövidebbek, szélesebb versenyzői kör számára érthetőek.

A fentiekkel azt szeretnénk bemutatni, hogy az egyszerű algoritmus végrehajtási képességen túl ezek a feladatok alkalmasak programozási ismeretek bevezetésére, emiatt a tanítási folyamatban hatékonyan felhasználhatók.

Számítógépes számonkérések

A magyar verseny második fordulójában három-öt kisebb, **konstruáló**, **szintetizáló** jellegű feladatot kell megoldani; a rendelkezésre álló idő 5 óra. Csak a futási eredményt értékeljük, nem pedig a megírt program szövegét. Súlyt helyezünk arra, hogy a versenyzők pontosan betartsák a specifikációt, ne csak tartalmilag, hanem formailag is legyenek előírás szerintiek az eredmények. Ebben a fordulóban korcsoportonként 200-300 versenyző szerepel.

Ez a feladattípus már lényegében megfelel a nemzetközi informatikai olimpiákon használt feladatoknak, van azonban több, lényeges különbség.

A korcsoportonként szervezett versenyben fokozatosan kell eljutni a diákolimpiák feladattípusaihoz.

A legkisebbeknek olyan feladatokat kell megoldani, amelyben néhány elemi adatból kell valamit kiszámítani.

Az időt (óra, perc, másodperc, századmásodperc) formában tároljuk. Olyan típusú kérdéseket szeretnénk feltenni, hogy ha most 8 óra 10 perc 12 másodperc 3 századmásodperc van, akkor mennyi lesz az idő 80 másodperc múlva, illetve hogy mennyi volt az idő ezelőtt 2 óra 15 perccel. Azaz az idő típusú adatokra el kell készíteni az összeadás és a kivonás műveletet.

Készíts programot, amely beolvas egy időpontot: O, P, MP, SZMP formában (0≤O≤23, 0≤P≤59, 0≤MP≤59, 0≤SZMP≤99), majd egy idő távolságot (0≤A, 0≤B, 0≤C, 0≤D) és kiírja, hogy mennyi volt az idő A óra B perc C másodperc D századmásodperccel korábban, illetve később!

Hasonló probléma (Third Millenium) található a XI. Lithuanian Olympiads in Informatics [9] olimpián a juniorok második fordulójában. E feladattípus jellemzője, hogy csupán a megoldás helyességével kell törődni, hatékonysági szempontokra itt még nem gondolunk.

A második és a harmadik korcsoportban részben elemi algoritmusokkal foglalkozunk.

Ennek fontos jellemzője, hogy a magyar informatika érettségi vizsgán is ilyen jellegű feladatok fordulnak elő. A verseny erre kiváló előkészítést jelent.

Az alábbi feladat egy intervallum-sorozat uniójának kiszámításáról szól.

Egy kiállítás három napon keresztül folyamatosan nyitva tart éjjel-nappal. A látogatóknak előre meg kellett venniük a jegyet, mégpedig úgy, hogy meg kellett mondaniuk, hogy mikor érkeznek, és mikor távoznak a kiállításról. A kiállítás szervezői így pontosan tudják, hogy mikor nem lesz senki a kiállításon. Azt tervezik, hogy csak azokra az időszakokra biztosítanak személyzetet, amikor lesz látogató.

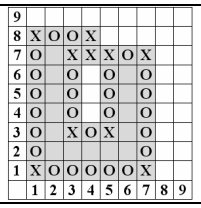
Készíts programot, amely kiszámítja azokat az időintervallumokat, amikor személyzetet kell biztosítani!

A másik feladattípus ismert, nevezetes algoritmusok alkalmazását igényli, az alábbi például egy terület befestését.

Egy négyzet alakú területre egy befestett sokszöget rajzoltunk, amelynek oldalai párhuzamosak a képernyő szélével.

Készíts programot, amely megadja, hogy hány képpontból áll a sokszög!

Az ábrán X jelöli a sarokpontokat, O a határvonalakat, s szürkére festettük a sokszöghöz tartozó összes pontot.



Ugyanilyen jellegű feladat például Horvátországból: SVEMIR (Croatian Highschool Competitions in Informatics, 2005, National Competition #1 [10]).

Több olyan feladat is előfordul, amelyek előkészítik a komolyabb algoritmusokat. Az alábbi feladat egy irányított gráffal kapcsolatos kérdéseket tesz fel, de közülük csak a harmadikhoz szükséges gráfbejárás, az első kettő tulajdonképpen adott tulajdonságú pontokat keres a gráfban.

Egy csapatversenyben N csapat vesz részt, a csapatokat 1 és N közötti sorszámukkal azonosítjuk. Ismerjük M mérkőzés eredményét, bármely két csapat legfeljebb kétszer játszhat egymással.

Írj programot az alábbi feladatokra!

A. Adj meg két csapatot, amelyek már legyőzték egymást!

B. Add meg azokat a csapatokat, amelyek már játszottak, de még senki nem győzte le őket!

C. Adj meg egy csapatot, amely „közvetve legyőzte magát” (azaz pl. A ilyen, ha A legyőzte B-t, B legyőzte C-t, ..., Y legyőzte Z-t és Z legyőzte A-t)!

Bár ezen feladatok némelyikénél a hatékonyság is számíthatna, ebben a fordulóban lényegében nem foglalkozunk ezzel. Az adatok mennyisége ugyanis vagy nagyon kicsi, vagy a legrosszabb megoldási ötletek esetén is viszonylag gyors megoldást kaphatunk. Ebben az esetben amely program 1 percen belül nem ad megoldást, az majdnem biztos, hogy 1 napon belül sem.

A harmadik fordulóban a feladatok típusa lényegében nem változik; a rendelkezésre álló idő 6 óra. Itt korcsoportonként 50-80 versenyző szerepel. A versenyzők itt is 1 perces időlimitet kapnak, ez azonban már alkalmas az exponenciális és a polinomiális algoritmusok megkülönböztetésére.

Itt már a 9-10. osztályosok feladatai is részben megfelelnek a diákolimpiákon előforduló feladatoknak, a 11-12. osztályosoké pedig csak ilyen típusú. Nem használjuk azonban még az összes lehetséges feladattípust. Jellemező erre a fordulóra a különböző gráf-algoritmusok használata, mint az alábbi, mélységi bejárásra épülő példából látszik:

A városi vidámpark több részlegből áll. Az egyes részlegeket kétirányú utak kötik össze. Az úthálózat olyan, hogy bármely részlegtől legfeljebb három közvetlen út vezet más részleghez, kivéve a főbejáratot tartalmazó részleget, onnan legfeljebb két másik részleghez vezet közvetlen út. Egy részleghez érve csak a részlegen keresztül lehet másik útra lépni. Minden részleghez el lehet jutni – esetleg más részlegeken keresztül – a főbejáratot tartalmazó részlegtől. Minden részlegbe csak az oda szóló belépőjeggyel lehet bemenni. Kedvezményesen lehet venni olyan belépőjegy köteget, amely minden részlegbe pontosan három jegyet tartalmaz.

Készíts programot, amely kiszámít egy olyan séta útvonalat, amely a főbejáratot tartalmazó részlegtől indul, oda ér vissza és minden részleget tartalmaz, de minden részleget legfeljebb háromszor!

Sok esetben nem gráf a feladat hátterében lévő adatstruktúra, hanem fa. Itt az alapvető egyszerű rekurzív bejárások mellett érdekes például a C részfeladat:

Egy titkos társaság hierarchikusan épül fel, minden tagja csak a felettesét és a hozzá közvetlenül beosztott legfeljebb két tagot ismeri. A társaságnak pontosan egy olyan tagja van, akinek nincs főnöke. Bármelyik tag küldhet levelet bármelyik tagnak. Azonban minden levél csak úgy juthat el a feladótól a címzetthez, hogy egy lépésben vagy a közvetlen főnökhöz, vagy közvetlen beosztotthoz továbbítódik.

Írj programot, amely adott két, X és Y tagra kiszámítja az alábbi kérdésekre adandó választ!

A. Hány beosztottja – nem csak közvetlen – van az X és az Y tagnak?

B. Hány lépéssel továbbítódik egy levél, ha X küld levelet Y-nak?

C. Mennyi a legkevesebb lépésszám, ami alatt biztosan odaér egy levél, bárki legyen is a feladó, illetve a címzett?

Minden évben szerepel a mohó stratégia, a fiatalabb korosztály általában olyan feladatot kap, amely emlékeztet valamelyik 1-2 évvel korábbi diákolimpiai, illetve olimpiai válogatóversenyes feladatra:

Egy népszerű zenekar a következő évre vonatkozó fellépéseit tervezi. Sok meghívása van fellépésre, ezek közül kell a zenekarnak választani, hogy melyeket fogadja el. Minden fellépés pontosan egy napot foglal el. Minden beérkezett meghívási igény egy ***(e, u)*** *szám-párral adott, ami azt jelenti, hogy az igénylő azt szeretné, hogy a zenekar olyan* ***k*** *sorszámú napon tartson nála koncertet, hogy* $e \le k \le u$*. A zenekarnak az a célja, hogy a lehető legtöbb fellépése legyen.*

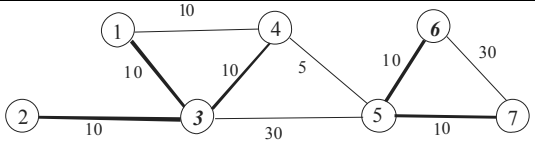
Készíts programot, amely kiszámítja, hogy mely meghívásokat fogadja el, hogy a következő évben a lehető legtöbb fellépése legyen, és a programod adjon is meg egy beosztást!

A diákolimpiai válogatóversenyen körülbelül 25-30 versenyző vesz részt, közülük kerül ki az IOI-s és a CEOI-s csapat 4-4 tagja. Alapelvünk, hogy az IOI-s csapatba 11-12. osztályos, a CEOI-s csapatba pedig 9-11. osztályos tanulók kerülhetnek, ezzel is szeretnénk lehetőséget biztosítani a fiatalabb korosztálynak a nemzetközi megmérettetésre.

E verseny feladattípusai, értékelési szempontjai, időlimitjei szó szerint azonosak az informatikai diákolimpiákkal. Határozott eltérés több diákolimpiától, hogy a feladatok jelentős részében van részfeladat, amelyre külön pontok szerezhetők. A tesztesetek kiválasztása is eltér a diákolimpiai feladatokétól. Erre mutat példát a következő, speciális minimális feszítőerdő megadását kérő feladat, és annak értékelése:

A Malomipari Vállalat K malomban őröl és csomagol lisztet. A lisztet N városba kell elszállítani úgy, hogy a szállítási összköltség a lehető legkisebb legyen.

Írj programot, amely megadja a minimális szállítási költséget, illetve minden városra, hogy oda melyik malomból kell szállítani a lisztet!



Értékelés:	
Minden városban van malom	*0+1 pont*
Sehol nincs malom	*1+0 pont*
Egyetlen malom, mindenhova van közvetlen út	*0+1 pont*
Egyetlen malom, több lépéses utak is vannak	*1+1 pont*
Egyetlen malom, nem a legrövidebb éleket kell választani	*1+1 pont*
Több malom, a legrövidebb éleket kell választani	*1+1 pont*
Több malom, nem a legrövidebb éleket kell választani	*1+1 pont*
Több malom, nem összefüggő gráf, minden komponensban van malom	*1+1 pont*
Több malom, nem összefüggő gráf, nem megoldható	*1+0 pont*
Véletlen közepes teszt	*1+1 pont*
Véletlen közepes teszt	*1+1 pont*
Véletlen nagy teszt	*2+2 pont*
Véletlen nagy teszt	*2+2 pont*

Az értékelésből látszik, hogy az egyszerű speciális esetek megoldását is pontozzuk. Másik jellemző, hogy elég sok pontot lehet szerezni a nem optimális megoldásokkal is, csak az utolsó 4 teszt különíti el az optimális megoldásokat (bár 12 pontot érnek a lehetséges 26-ból).

A továbbiakban csak az új feladattípusokat nézzük át. Az elmúlt években az informatikai diákolimpiákon gyakran megjelennek a kombinatorikus feladatok, emiatt az olimpiai válogatásunkon is szükség van rá:

A processzorgyártó cégek megállapodtak abban, hogy milyen rendszert alkalmaznak az általuk gyártott processzorok azonosítására. Minden cég kap egy betűkészletet, és ezekből kell az azonosító kódot képeznie úgy, hogy minden betű pontosan egyszer szerepeljen az azonosítóban. Például egy cég azt kapta, hogy minden azonosítója egy-egy 'a', 'b', 'c', 'd' és 'x' betűt tartalmazzon. A processzorok a gyártási sorrendben ábécé szerint növekvően kapják mindig a következő azonosítót.

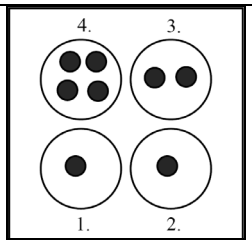
Készíts programot, amely adott azonosítóra kiszámítja a következő két kérdésre a választ.

A: Hányadik a gyártási sorrendben a processzor? (0-tól sorszámozva)

B: Mi az ábécé sorrendben rákövetkező szabályos azonosító?

Szintén jellemző feladatok a diákolimpiákon az interaktív feladatok osztályába tartozó kétszemélyes játékok. Ilyet általában a válogatás utolsó fázisában használunk:

A Mancala családba tartozó játékok, amelyeket kavicsokkal és üregekkel játszottak, a legősibbek közül valók. A mini mancala változatában, amely kétszemélyes játék, négy üreg van. Az 1. és a 2. üreg az egyik, a 3. és 4. üreg a másik játékoshoz tartozik. A játék kezdetén az üregekbe véletlenszerűen beraknak legfeljebb 8 kavicsot. A játékosok felváltva lépnek, az 1-es játékos kezd. A soron következő játékos kiválaszt a hozzá tartozó üregek közül egyet, majd kiveszi az abban lévő összes kavicsot. Ezután a kivett kavicsokból egyet eldob, a többit pedig szétosztja az üregek között az alábbiak szerint. Az órajárással ellentétes irányban haladva minden érintett üregbe rak egy kavicsot, de kihagyja azt az üreget, amelyből kivette a kavicsokat. Például, ha az 1. üreget választja, amiben 6 kavics van, akkor sorrendben a 2., 3., 4., 2. és 3. üregbe rak egy-egy kavicsot.



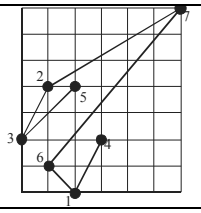
A játék akkor ér véget, ha a soron következő játékos nem tud lépni, mert mindkét hozzá tartozó üreg üres. Az a játékos nyer, aki utoljára tudott lépni.

Írj programot, amely a játékot kezdő 1. játékos nyerő stratégiáját valósítja meg!

Az elmúlt 2-3 évben jelentek meg hangsúlyozottan a geometriai feladatok, amelyek azonban a diákolimpiai szokásoknak megfelelően csak egész aritmetikára épülve is megoldhatók:

*Adott a síkon egy **P** ponthalmaz és két kitüntetett pontja; **a** és **b**. Kiszámítandó egy olyan **a**-ból **b**-be vezető, nem-metsző törtvonal, amelynek csúcspontjai pontosan a **P** ponthalmaz elemei. A pontokat az **1,…,N** számokkal azonosítjuk. Egy ilyen törtvonal megadható a pontok azonosítóinak egy olyan felsorolásával, amelyben az első elem **a**, az utolsó **b**, továbbá az egymást követő pontokat kötjük össze egyenes szakaszokkal.*

*Írj programot, amely kiszámít egy **a**-ból **b**-be vezető, nem-metsző törtvonalat!*



Összefoglalóan nézzük meg, milyen feladattípusokat használunk az egyes korosztályok versenyein:

5-8. osztály	Ciklus nélküli feladatok, számlálás, keresés, rendezés, kiválogatás, maximumkiválasztás, unió, metszet, szimuláció, szövegelemzés, szövegformázás, speciális rendezések, egyszerű online algoritmusok, kis elemszámú sorozatok feldolgozása.
9-10. osztály	Az előző feladattípusokon túl: Geometriai algoritmusok, intervallumokkal, halmazokkal kapcsolatos algoritmusok, Rekurzió, fákkal kapcsolatos rekurzív algoritmusok, gráfok elemzése, elemi gráfalgoritmusok, backtrack, grafikai alapalgoritmusok, mohó stratégia, dinamikus programozás, adatstruktúrák alkalmazása, szövegformázás, szövegelemzés, bitminták elemzése, szimuláció, speciális struktúrák bejárása, online algoritmusok, feladatmegoldás automatákkal.
11-12. osztály	Az előző feladat-típusokon túl: Gráfbejárás, feszítőfák, szövegfeldolgozás (tömörítés, keresés), automaták, kombinatorikus algoritmusok.
Olimpiai válogatóverseny	Az előző feladat-típusokon túl: Kombinatorikus algoritmusok, interaktív feladatok, kétszemélyes játékok nyerő stratégiája, feladatmegoldás speciális „gépekkel”.

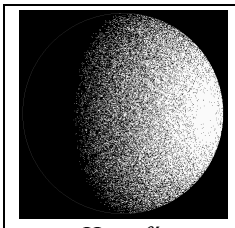
A fenti táblázatból látszik, hogy a feladattípusokban nagy ugrás az első két korcsoport határán van. A többi korcsoportban, illetve versenyeken a feladatok témakörei azonosak, csupán a konkrét feladatok nehézsége különböző. Például a 9-10. osztályosok gráfokkal kapcsolatos algoritmusaiban a legbonyolultabb egy egyszerű gráfbejárás. Az egyszerűbb gráfos feladataik a gráfok elemzésével megoldhatók (pl. egy pontnak hány bemenő éle van, hány kimenő éle van, elágazásmentes láncok felfedezése a gráfban, …). A 11-12. osztályosoknál ezzel szemben már a legegyszerűbb feladatnál is szükség van gráfbejárásra, minimális költségű feszítőfa előállítására, … A válogatóversenyen pedig ugyanezen feladattípusok esetén sokkal fontosabb a hatékony megoldás, a megfelelő adatstruktúra választása, …

Vannak ezen kívül speciális versenyek, ahol a feladattípusok is eltérhetnek a fentiektől.

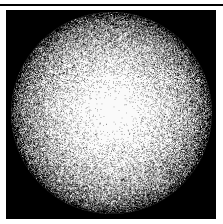
3. IZSÁK IMRE GYULA MATEMATIKA-FIZIKA-SZÁMÍTÁSTECHNIKA VERSENY

Ezen a versenyen mind a három tantárgy a saját feladatain túl olyan feladatokat is ad, amelyek a másik tantárgyhoz kapcsolódnak. Így szerepelt például olyan fizikai feladat, amelyben a fizikai tudás alapján kellett meghatározni egy CD maximális lehetséges kapacitását.

Az informatikai feladatok egy része a fizikához, más része a matematikához kapcsolódik. Ilyen például a következő feladat:



Egy gömböt úgy tudunk a síkban ábrázolni, hogy a felénk forduló részét fényesebbre festjük, mint a kevésbé felénk fordulókat. A fényesség a fény beesési szögének koszinuszával arányos. A fényességet úgy állítjuk be, hogy a legfényesebb helyeken nagyon sűrűn teszünk a sötét háttérre fehér pontokat, s minél kisebb a fényesség, annál ritkábban. Ha a gömböt a nézőpontból egy párhuzamos fény-nyalábbal világítjuk meg, akkor a jobboldali ábrán látható képet kapjuk.



Ha a fény-nyalábot a nézőponthoz képest az y-tengely körül 60 fokkal elforgatjuk jobbra, akkor a baloldali ábrán látható képet kapjuk.

Készíts programot, amely beolvassa a fény-nyaláb és a nézőpont által bezárt szöget, majd kirajzolja a megvilágított gömböt úgy, hogy a határvonalát piros körrel rajzolja!

4. INFORMATIKAI SZAKKÖZÉPISKOLÁSOK SZAKMAI TANULMÁNYI VERSENYE

Ezen a versenyen a többiektől határozottan eltérő feladatok fordulnak elő. Ezek jellege erősen gyakorlati, viszonylag egyszerű algoritmusokat kell használni egy-egy komplex feladat megoldásában. Itt ezért fontosabb a feladatok jó felbontása részfeladatokra, majd a részfeladatok megoldásainak helyes összeépítése.

A következő feladat tulajdonképpen nagyon hasonlít a diákolimpiák interaktív feladataira. Egy adatsorozaton kell egy adott ablakot csúsztatni, miközben a csúsztatás szabályait a menet közben bármikor érkező műveletek módosíthatják. Ezen a versenyen kézi értékelés volt, a képernyőn megjelenő tartalom alapján. Az automatikus értékelés a kimenet célszerű megfogalmazásával kicsit nehezen, de megoldható.

Készíts programot egy N vágányt tartalmazó vasútállomás elektronikus információs táblájának kezelésére! Az információs táblán jelenjen meg az állomás neve, az aktuális idő, valamint N sorban az induló és érkező vonatok adatai kétféle változatban:

- *az állomásról induló vagy oda érkező következő N db vonat indulási, illetve érkezési idő szerint sorbarendezve, közöttük azonban csak olyanok szerepelhetnek, amelyek ideje legfeljebb X órával nagyobb az aktuális időpontnál;*
- *az N vágányon, vágányonkénti sorrendben a következő induló vagy érkező vonat, de itt is csak olyanok szerepelhetnek, amelyek ideje legfeljebb X órával nagyobb az aktuális időpontnál.*

A kétféle megjelenítés között a program felhasználója választhat tetszőleges időpontban.

A tábla kezeléséhez szükséges adatokat a vasútvonal teljes menetrendjét tartalmazó állományban tároljuk.

Bármikor érkezhet az adott időpont után érkező vonatokra vonatkozó üzenet – lehetséges, hogy a menetrendhez képest késnek. Az állomásfőnök bármikor megváltoztathatja a vonatok indulási idejét, illetve érkezésüket megelőzően a hozzájuk rendelt vágánysorszámot.

Készíts programot az információs tábla kezelésére!

5. ÖSSZEFOGLALÁS

Reméljük, a példákkal sikerült igazolni az informatikai versenyek lehetséges sokszínűségét. Úgy gondoljuk, hogy szerepe lehet mind a számítógép nélküli, algoritmusokról szóló versenyeknek, versenyfordulóknak, mind pedig a számítógépet használó versenyeknek.

Mindkét fajta versenynek vannak olyan részei, amelyekhez szükséges tudással minden tanulónak rendelkezni kell. Az országos versenyek első, illetve részben a második fordulójában ilyen tudást célszerű mérni.

A legtehetségesebbek számára a verseny már az informatikus pályára való előkészületeket jelenti, így nekik már a diákolimpiai feladatokhoz hasonlókkal kell találkozniuk.

Fontosnak tartjuk az informatikai ismeretek bővítésében is a fokozatosságot, s ezt a példáinkkal igazoltuk.

KÖNYVÉSZET

[1.] *http://tehetseg.inf.elte.hu/* – a magyarországi informatika versenyek kiinduló honlapja.
[2.] *http://www.math.bas.bg/bcmi/* – Bulgarian Competitions in Mathematics and Informatics.
[3.] *http://www.ksp.sk/* – Slovak National Olympiad in Programming.
[4.] *http://www.bwinf.de/* – a németországi informatika versenyek kiinduló honlapja.
[5.] *http://olympiads.win.tue.nl/ioi/index.html* – International Olympiad in Informatics.
[6.] *http://ceoi.inf.elte.hu* – Central-European Olympiad in Informatics.
[7.] *http://www.math.bas.bg/bcmi/noi85.html* – First Bulgarian National Olympiad, problems.
[8.] *http://olympiads.win.tue.nl/ioi/ioi95/contest/print.htm*l – IOI95 számítógép nélküli feladat.
[9.] *http://ims.mii.lt/olimp/?lang=en&sk=pasirengimas&id=0610* – Lithuanian Olympiad in Informatics, tasks.
[10.] *http://www.hsin.hr/2005/* – Croatian Highschool Competitions in Informatics, 2005.
[11.] *http://www.amt.edu.au/aicsample.html* – Australian Informatics Competition: Sample Questions, 2005.