SUDOKU-algoritmusok összevetése, avagy SOLVER vs. egyedi kód

(Comparing of SUDOKU-algorithms or SOLVER vs. customized source code)

Pitlik László, Pitlik Marcell – MY-X team

Kivonat: A programozni tanulás, hasonlóképpen a jogalkotáshoz, ahol létezik a jogfilozófia jelensége, szintén elvárja, hogy a program fogalmát az operatív végrehajtástól néhány lépéssel hátrébb lépve is értelmezni tudjuk. Ha metaszinten minden program, ami bármilyen korábbi manuális tevékenységet képes kiváltani, vagyis képes egy adott feladat végrehajtásának hatékonyságát – az egy időegység alatt előállítható hasznosságot – növelni, akkor kijelenthető, hogy az egér/billentyű-aktivitások rögzítésétől, az Excel-számolási sémáin át pl. a Solver és/vagy kimutatásvarázslás folyamatokba való bevonásáig minden egy fajta program, nem csak a klasszikus programnyelvekhez kötődő programok. Jelen cikk célja rámutatni arra, miben más a Solver-alapú SUDOKU-megoldás a hagyományos programtervezéssel/programozási logikákkal összevetve.

Kulcsszavak: szoftvertervezés, szoftvertesztelés, algoritmus-tervezés, algoritmus-tesztelés

Abstract: The philosophy of law is an existing abstraction parallel or rather about the operative legislation processes. Therefore, it should not be strange, if it will be spoken about the philosophy of creating/planning of source codes. On a meta level, each phenomenon which is capable of increasing efficiency (it means utility per time unit) of the programming through substitutions of manual activities, may be called as a source code. So, a keylogger (the logical series of mouse/keyboard-activities may also be seen as a program or even the Excel-calculation schemes with or without Solver-modules and/or pivot-reports may be interpreted as a kind of support mechanisms for increasing efficiency. Therefore, not only the classic programming languages can ensure source codes. This article demonstrates - based on a SUDOKU-task – what kind of differences can be identified if the classic and the Solver-based approaches will be compared.

Keywords: software planning, software testing, algorithm planning, algorithm testing

Bevezetés

A programozás tanítására számos stratégia képzelhető el. A legegyszerűbb – tanári oldalról, ha hagyjuk az autodidakta fejlődést realizálódni. A másik quasi szélső helyzet, ha minden, amit csak eddig egy adott programnyelv kapcsán említésre érdemesnek gondoltak a már programozni tudók (vö. egy fajta szuper-fórum), valamilyen sorrendben (esetleg akár egyéni bejárást /is/ megengedve) megosztásra kerül az éppen tanulni vágyókkal. Míg az első közelítés nem való a tapasztalatok alapján vélhetően mindenkinek, addig a másik ismét csak a tapasztalatok alapján zömmel lassú, nem hatékony, unalmas, esetleg részlegesen felesleges erőforrás-lekötéseket generáló is. A két megközelítés között helyezkedik el a gamifikált megoldások részhalmaza: ebben a közelítésben a konduktor érdekes/játékos feladatok kapcsán várja el az öntanulást úgy, hogy közben egymástól akár nagyon távol álló filozófiákat is bemutat összehasonlító jelleggel, ezzel is katalizálva a programozásról a kiképzendőben spontán fejlődő meta-szintű erőterek arányait.

Ebben a cikkben egy csak és kizárólag SUDOKU-megoldást támogató algoritmus-fejlesztés folyamata kerül szembe állításra a SOLVER alapú (közel univerzális probléma-kezelő motorra alapozó) megoldás-tervezéssel.

Előzmények

Az alábbi dokumentumok a programozás és/vagy Excel-tanulás lehetséges stratégiáinak egy-egy részhalmazát mutatják be eltérő aspektusokból:

- https://miau.my-x.hu/miau/253/cipher1.docx
- https://miau.my-x.hu/miau/253/cipher2.docx
- https://miau.my-x.hu/miau/253/cipher3.docx - ill. https://miau.my-x.hu/miau/253/cipher123.docx
- https://miau.my-x.hu/miau2009/index.php3?x=e0&string=hangos
- https://miau.my-x.hu/digeco/coco_prg1.xlsm
- https://miau.my-x.hu/digeco/coco_js.docx ill. https://miau.my-x.hu/digeco/coco_js.html
- https://miau.my-x.hu/miau/solver4u/
- …

A SUDOKU-algoritmus

Ebben a cikkben nem a konkrét algoritmusok, hanem sokkal inkább az ezeket előkészítő specifikációk kerülnek bemutatásra, hiszen ahogy azt az előzmények kapcsán világosan látni lehetett, a tételes kódírás nagyon eltérő megoldásokhoz vezethet.

Elsőként válasszunk egy egyszerű, áttekinthető, de már jövőbe mutató (általános érvényűséget súroló) SUDOKU-rejtvényt: https://miau.my-x.hu/miau/solver4u/sudoku_excel_solver.xlsx

x1	4	2	x2
2	x3	x4	x5
x6	x7	x8	1
x9	1	3	x10

1. ábra: A rejtvény (forrás: saját ábrázolás)

Maga a feladat triviális: minden sorban és oszlopban minden engedélyezett számjegy (1-2-3-4) csak egyetlen egyszer fordulhat elő!

Egy lehetséges megoldási stratégia lépései (L_i):

- L1: Ellenőrizzük le van-e a kiindulási állapotban olyan sor és/vagy oszlop, melyekben csak egyetlen egy pozíció üres. Vagyis rendeljünk hozzá minden sorhoz és oszlophoz egy dinamikusan módosuló tartalmú változót/tömbelemet, melyek értéke adott pillanatban az adott sorban/oszlopban lévő nem üres cellák száma. Így ezek értékkészlete 0-1-2-3-4, ahol a 0 azt jelenti, hogy nincs üres cella a sorban/oszlopban, vagyis minden érték adott már. A 4 azt jelenti, hogy létezik olyan sor/oszlop, mely teljesen üres.
- (Ha feltételezzük egy pillanatra, hogy olyan játék meg sem oldható, melyben van legalább egy sor/oszlop, mely teljesen üres, akkor a fenti 4-es érték léte egyben azonnali kilépési feltétel is lehetne azzal az üzenettel, miszerint: pl. „a feladvány nem oldható meg egyértelműen”)
- (Hasonlóképpen kilépési feltétel lehetne, ha egy sorban/oszlopban már a kiindulási állapotban min. két azonos számjegy szerepel – hiszen ilyenkor az üzenet az lehetne, hogy pl. „a feladat téves kiindulási adatokat tartalmaz”)

- L2: Az L1 tömb ismeretében akkor lehet azonnal egy/több pozícióba (cellába) értéket definiálni, ha a tömbben legalább egy darab 1-es érték szerepel. Ha több is szerepel, akkor véletlenszerű sorrendben lehet ezeket sorban kezelni kezdeni.
- L3: Az első azonnal megadható cella koordinátáinak ismeretében, melyek vagy tárolva vannak a tömbben magában, vagy következnek a tömb szerkezetéből, azonnal le lehet ellenőrizni/ki lehet számolni, melyik az a számjegy, ami még nem szerepel és így meg is van az első becslés/tipp.
- L4: a friss tipp alapján a dinamikusan újra számolódó tömb értékeit frissíteni kell
- L5: amint egy új cella feltöltésre került, akkor a folyamat vissza kell, hogy ugorjon az L1-re és mindaddig önmagán belül kell forognia a folyamatnak, amíg 3-as érték van az egyre frissülő tömbben…
- L6: ha az L1:L5 lépéssor konklúziója az, hogy nincs további 3-as érték a tömbelemek között a frissítések kapcsán sem, akkor a folyamat átlép a triviális megoldási stratégiából egy következő (komplexebb) stratégiai szintre, ahol minden egyes üres cella kapcsán egy új (ismét csak majd dinamikusan változó tartalmú) tömb leírja, mennyi a még lehetséges megoldások száma (vö. 0 ← ha megvan a becslés/alapadat, 1 ← ha nem lenne az L1:L5 folyamat, akkor lehetne ilyen érték (vö. következő megjegyzés), 2 vagy 3 vagy 4 ← ha az L1:L5 folyamat kapcsán sem sikerült előbbre jutni)
- (Ha tehát az új tömbben 1-es érték fordulna elő az L1:L5 után azonnal, akkor valami zavar van a programlogikában, mert az új tömb 1-es értéke azonos az első tömb 3-as értékének értelmével, ahol az első tömbben már nem maradthatott 3-as érték az L1:L5 folyamat végére, vagyis ez is egy önellenőrzési feltétel, ami nagyon fontos, hogy minél több legyen a teljes folyamatban).
- L7: Ha az L6 eredményei között van 2-es érték (és elvárás, hogy ennek pozíciója egyértelműen adott legyen a lehetséges értékek darabszáma mellett ezek pontos megnevezésével együtt), akkor a még becslést elváró cellák kapcsán szóba jöhető számjegyek közül az eleve csak 2 lehetőséget felmutató cellák közül véletlenszerűen választott cellánként ki kell zárni az érintett sorok/oszlopok teljes tartalmának figyelésével azokat, melyek már az adott sorban és/vagy oszlopban szerepelnek BÁRHOL – vagyis az új tömb frissítése két lépcsős: az elem-kizárások után a fennmaradó potenciális elemszám is újra kalkulálandó.
- L8: S amint az új tömbben 1-es érték jelenik meg akár csak egyetlen egyszer is, akkor a vezérlés azonnal vissza kell, hogy ugorjon az L1:L5 ciklusba… és csak akkor kerül ismét az L6-ra a vezérlés, ha minden L1:L5 tipp a megfelelő módon előállt…
- L9: A fenti lépéssorról különösebb bizonyítás nélkül belátható, hogy érdemi esélye lesz sokféle rejtvény megoldásának, de az is vélelmezhető, hogy nem minden számmisztika esetén elegendő a már kezelt komplexitás – vagyis L9-ágra akkor kerül a vezérlés, ha nem lehet visszalépni az L6:L8-ból az L1:L5-be, mert minden még üres cellára igaz, hogy legalább két potenciális elemet kellene még értelmezni valahogy – ahol az L9-re kerülés lehet annak is a jele, hogy a feladat téves, vagy annak, hogy a megoldás még nem elég komplex…

Az alábbiakban következzék a választott példa tételes értelmezése a fenti (L1:L9) stratégiai vázlat alapján – nem feltétlenül követve tételesen az előírt elveket egy fajta alternatív gondolkodásmód demonstrálásaként, ahol az egyes lépések során az előzmények alapvető befolyással bírnak a következő lépések értelmére, tartalmára:

- első megállapítás, hogy az első L1:L5 folyamat eredménytelenül zárul az 1. ábra alapján, vagyis tipp még nem keletkezett, de már az L6-nál vagyunk…
- x1:
    - lehet 1 vagy 3 az első (felső, saját) sor alapján
    - (nem lehet 2 a második (felülről) sor ill. a bal szélső oszlop alapján, de ez nem segít előbbre jutni)
        - így x1 esetében a szűkítés sikertelen
        - x1 kapcsán nem születhet tipp, vagyis az 1. ábra egyelőre még változatlan
        - s x1 egyben demonstrálja, hogy az előrelépés nem kényszerű (melyre x5 és x4 kapcsán is láthatunk még példát)
- x2:
    - mivel x1 vizsgálata nyomán az új tömb semmiben nem változott, így lehet 1 vagy 3 az első (felső, saját) sor alapján
    - nem lehet 1 a harmadik sor/jobb szélső oszlop alapján
        - vagyis a szűkítés sikeres
        - x2=3 (az 1. ábra változik)
        - az átlépés az L1:L5 folyamatba azonnal is lehetséges, ami azonnal tippé konvertálná x1 értékét
        - vagy az új tömb jelzi, hogy már legalább egy pozíció okán az átlépés az L1:L5-be az összes szűkítési kísérlet után szükségszerű lesz
        - itt és most az azonnali átlépéses verzió alapján x1=1 (az 1. ábra változik)
        - (belső ellenőrzési pont: felső sor számjegyeinek összértéke = 10)

x1	4	2	3
2	x3	x4	x5
x6	x7	x8	1
x9	1	3	x10

1	4	2	3
2	x3	x4	x5
x6	x7	x8	1
x9	1	3	x10

2. ábra: az első és a második tipp hatásai (forrás: saját ábrázolás)

- x3:
    - lehet 2 vagy 3 a balról második (saját) oszlop alapján
    - nem lehet 2 a felülről második (saját) sor alapján
        - a szűkítés sikeres
        - x3 = 3
        - x7 = 2

1	4	2	3
2	3	x4	x5
x6	2	x8	1
x9	1	3	x10

3. ábra: a harmadik és a negyedik tipp hatásai (forrás: saját ábrázolás)

- x4:
  - lehet 1 és 4
  - további szűkítés nem lehetséges
- x5:
  - lehet 2 és 4 a jobb szélső (saját) oszlop alapján
  - nem lehet 2 a felülről második (saját) sor alapján
    - szűkítés sikeres
    - x5 = 4 (láncreakció indul)
    - x4 = 1
    - x8 = 4
    - x6 = 3
    - x9 = 4
    - x10 = 2
- feladat megoldva (minden sor és oszlop összege legyen 10 az önellenőrzés keretében)

1	4	2	3
2	3	1	4
3	2	4	1
4	1	3	2

4. ábra: a megoldás (forrás: saját ábrázolás)

A SOLVER-alapú megoldás

A Solver-alapú gondolkodásmód nem tipikus része a matematika és az informatika/számítástechnika oktatásának – ill. interdiszciplináris jelleggel sem szerves része semmilyen közismert tantervnek. A személyes felső tagozatos, középiskolás, HHH, felnőtt-képzési oktatási tapasztalatok alapján elmondható, hogy a Solver-alapú, ill. általában véve az Excel-alapú (pl. erőből politizáló) problémamegoldás elhanyagolása a klasszikus didaktikákhoz képest komoly hatékonysági hátrányt és motivációvesztést okoz generációról generációra.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
1											1	2	3	4		1	2	3	4
2	1	4	2	3	10		1.0	3.0	1.0		1	1	1	1	1	1	1	1	1
3	2	3	1	4	10		3.0	1.0	4.0		1	1	1	1	2	1	1	1	1
4	3	2	4	1	10		3.0	2.0	4.0		1	1	1	1	3	1	1	1	1
5	4	1	3	2	10		4.0	2.0	1.0		1	1	1	1	4	1	1	1	1
6	10	10	10	10	0.00		27.0								0				
7							0												

5. ábra: A Solver-alapú megoldás áttekintő képe (forrás: saját ábrázolás)

Az 5. ábra egy Solver-alapú SUDOKU-megoldó keretrendszer felületét vagyis a hiba-számításig vezető utat mutatja be (rendszerhiba=G7-cella).

A G-H-I oszlopok színes celláira azért van csak szükség, hogy a Solver számára a változó cellák egy közös tömbként kijelölhető helyre kerüljenek. S a narancssárga cellák értéke lényegrelen.

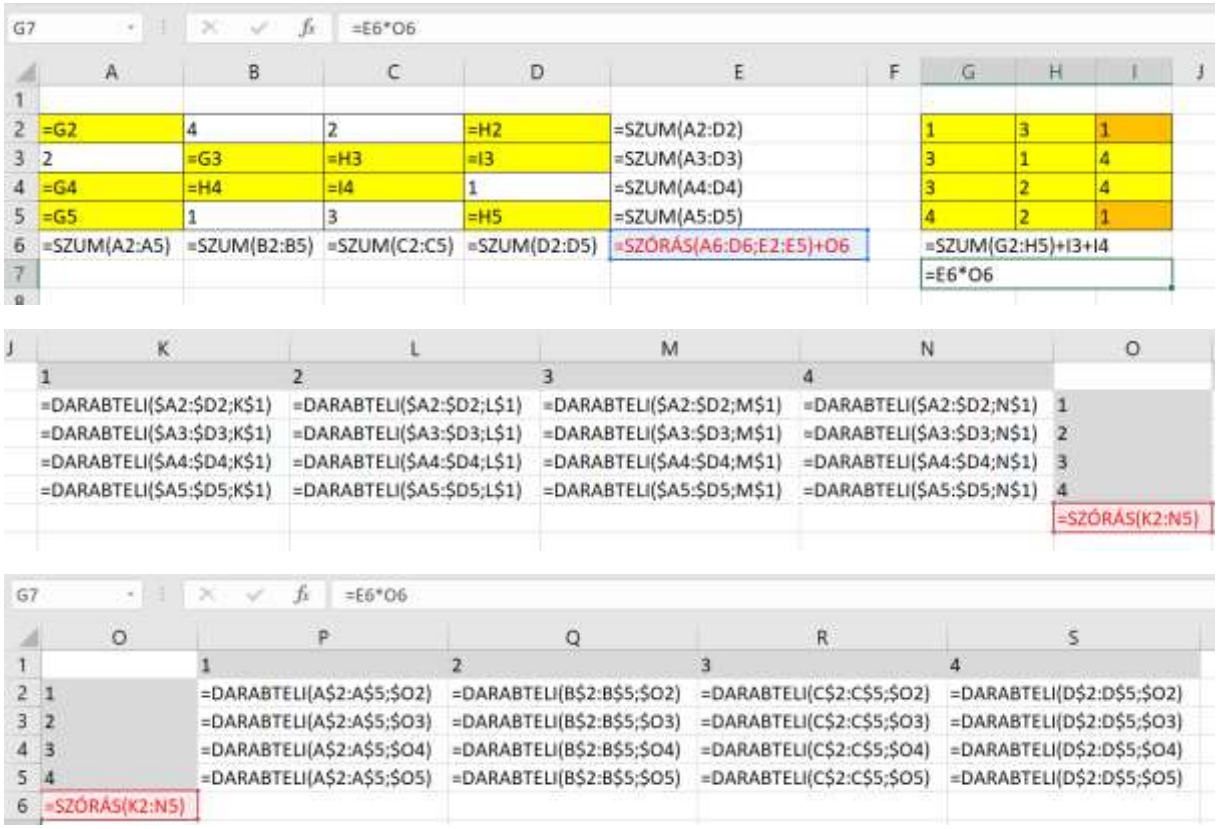
Az A-B-C-D oszlopok sárga cellái jelzik a G-H-I színes cellákból átvett megoldás-elemeket.

Az E oszlop és a 6. sor az önellenőrzés rétegei.

A K-L-M-N oszlopok és a P-Q-R-S oszlopok a sorokban és oszlopokban elhelyezkedő 1-2-3-4 számjegyek darabszámait mutatják.

A hiba fogalma komplex: egyrészt a számjegyek elemszámainak szórása, másrészt a sor/oszlop-ellenőrző összegek szórása kerül egymással összeszorzásra – a cél a nulla érték!

A 6. ábra a fenti értelmezések operatív (képlet-szintű) nézete cellánként.

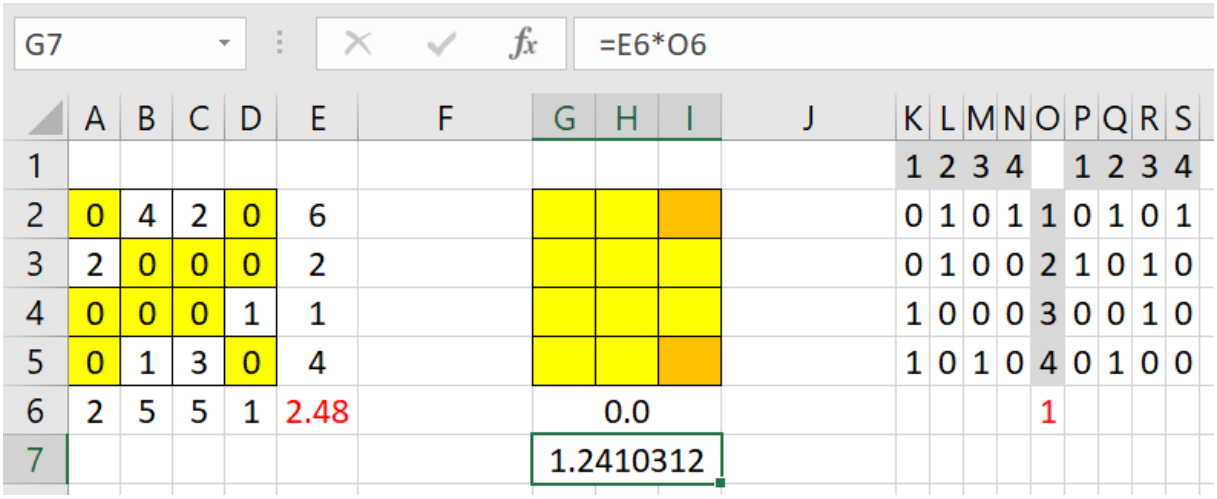


	A	B	C	D	E	F	G	H	I	J
1										
2	=G2	4	2	=H2	=SZUM(A2:D2)		1	3	1	
3	2	=G3	=H3	=I3	=SZUM(A3:D3)		3	1	4	
4	=G4	=H4	=I4	1	=SZUM(A4:D4)		3	2	4	
5	=G5	1	3	=H5	=SZUM(A5:D5)		4	2	1	
6	=SZUM(A2:A5)	=SZUM(B2:B5)	=SZUM(C2:C5)	=SZUM(D2:D5)	=SZÓRÁS(A6:D6;E2:E5)+O6		=SZUM(G2:H5)+I3+I4			
7							=E6*O6			

	K	L	M	N	O
	1	2	3	4	
	=DARABTELI($A2:$D2;K$1)	=DARABTELI($A2:$D2;L$1)	=DARABTELI($A2:$D2;M$1)	=DARABTELI($A2:$D2;N$1)	1
	=DARABTELI($A3:$D3;K$1)	=DARABTELI($A3:$D3;L$1)	=DARABTELI($A3:$D3;M$1)	=DARABTELI($A3:$D3;N$1)	2
	=DARABTELI($A4:$D4;K$1)	=DARABTELI($A4:$D4;L$1)	=DARABTELI($A4:$D4;M$1)	=DARABTELI($A4:$D4;N$1)	3
	=DARABTELI($A5:$D5;K$1)	=DARABTELI($A5:$D5;L$1)	=DARABTELI($A5:$D5;M$1)	=DARABTELI($A5:$D5;N$1)	4
					=SZÓRÁS(K2:N5)

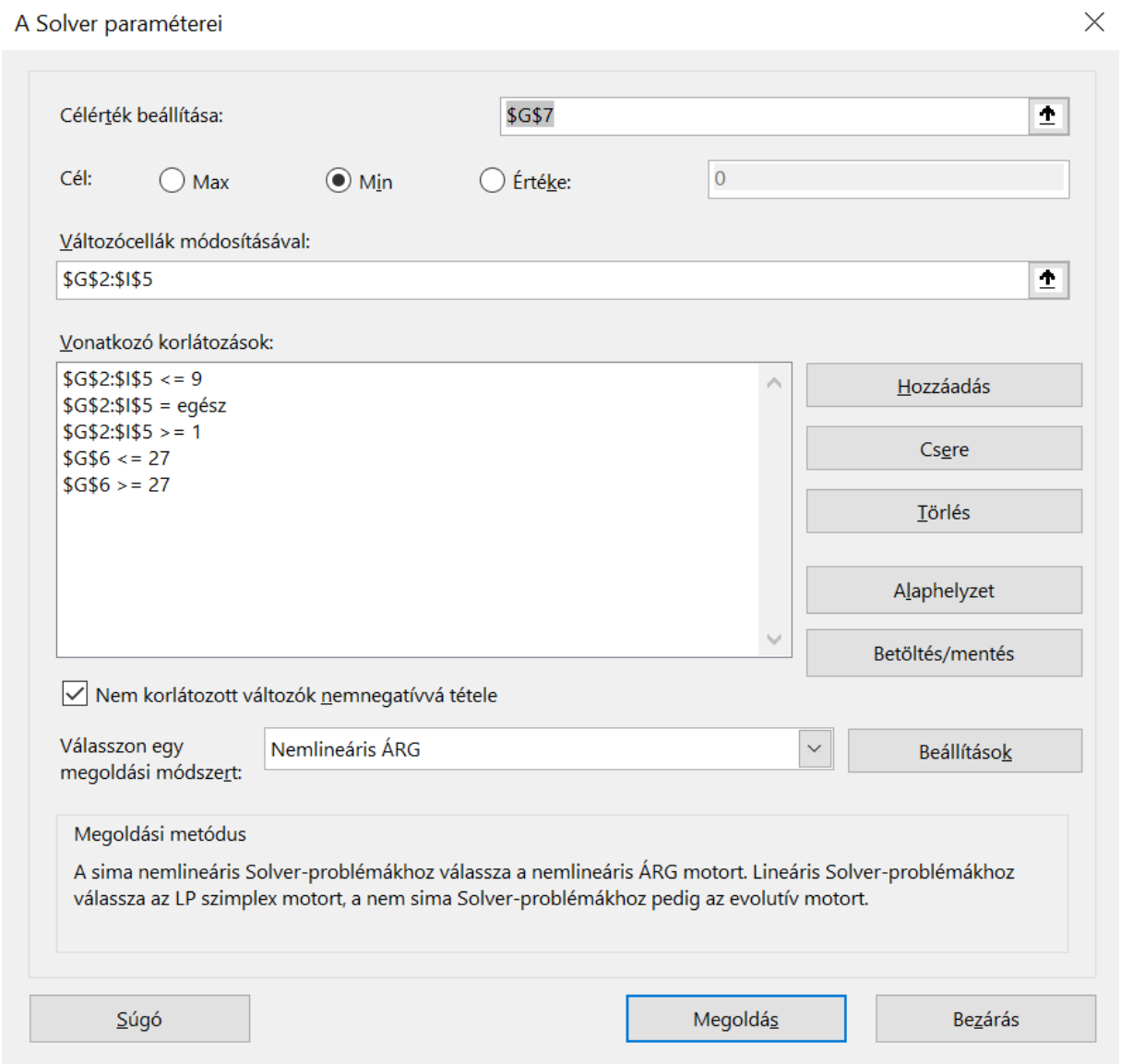
	O	P	Q	R	S
1		1	2	3	4
2	1	=DARABTELI(A$2:A$5;$O2)	=DARABTELI(B$2:B$5;$O2)	=DARABTELI(C$2:C$5;$O2)	=DARABTELI(D$2:D$5;$O2)
3	2	=DARABTELI(A$2:A$5;$O3)	=DARABTELI(B$2:B$5;$O3)	=DARABTELI(C$2:C$5;$O3)	=DARABTELI(D$2:D$5;$O3)
4	3	=DARABTELI(A$2:A$5;$O4)	=DARABTELI(B$2:B$5;$O4)	=DARABTELI(C$2:C$5;$O4)	=DARABTELI(D$2:D$5;$O4)
5	4	=DARABTELI(A$2:A$5;$O5)	=DARABTELI(B$2:B$5;$O5)	=DARABTELI(C$2:C$5;$O5)	=DARABTELI(D$2:D$5;$O5)
6	=SZÓRÁS(K2:N5)				

6. ábra: Az 5. ábra mögötti képletek (forrás: saját ábrázolás)



7. ábra: Az üres feladat Solver-nézete (forrás: saját ábrázolás)

A 7. ábra mutatja be a Solver indítása előtti állapotot. A 8. ábra a Solver paramétereit tárja fel:



8. ábra Solver-paraméterek (forrás: saját ábrázolás)

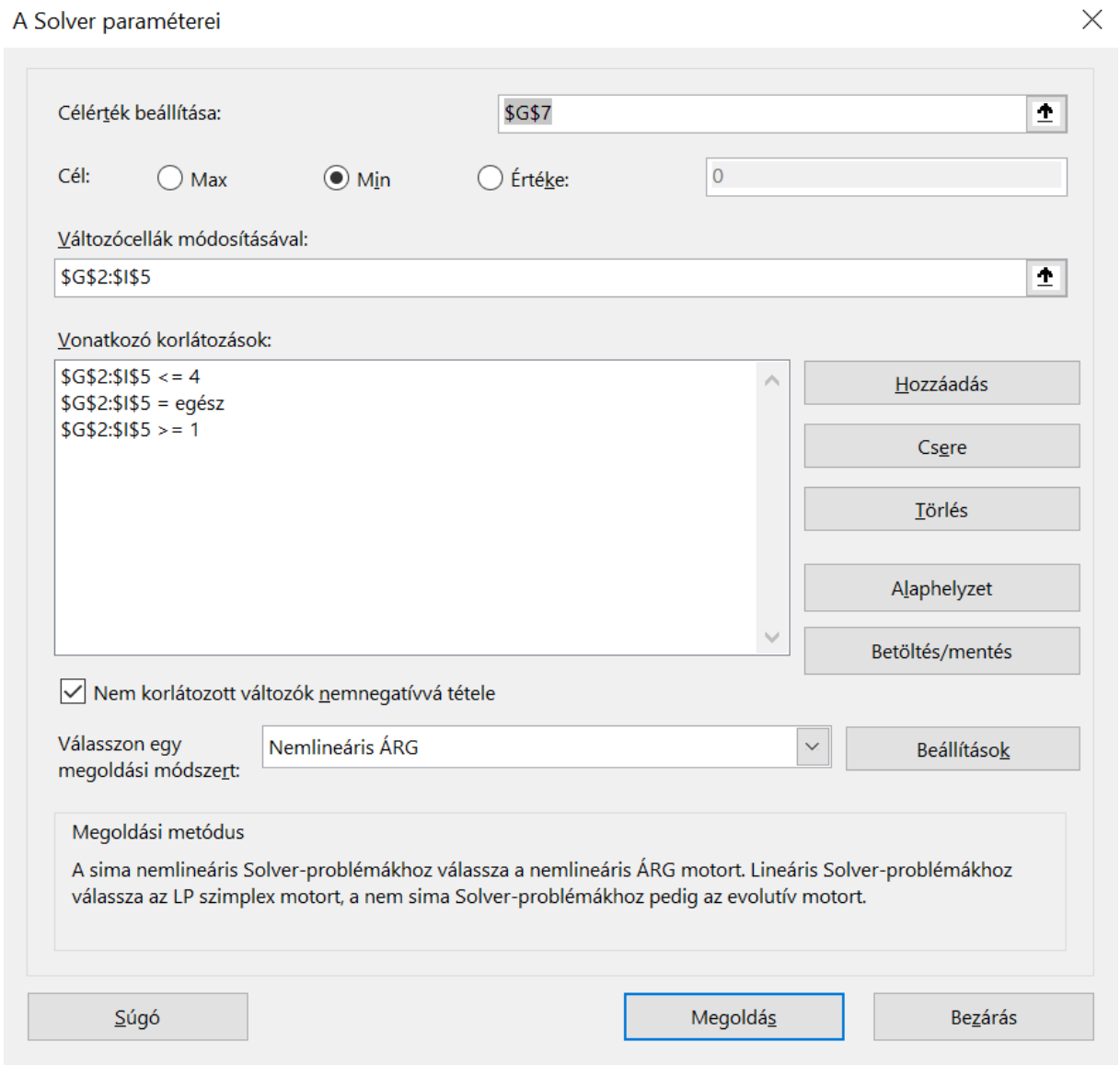
A Solver-paraméterek közül az egy-jegyűség (<=9) helyett állhatna a „<=4” paraméter is.

A G6 = 27 feltételt kép korlátozó feltétel tudja kikényszeríteni, ami egyben felelős azért, hogy a legkisebb számjegyekkel dolgozzon a rendszer.

A 10. ábra mutatja be a Solver-t csökkentett korlátozó feltétel-listával, s gyorsabb futással (vö. 21 iteráció).

9. ábra: A Solver futás alatti üzenetei (forrás: saját ábrázolás)

A 9. ábra alapján belátható, hogy a Solver sok-sok részletszámítás alapján jut el a megoldáshoz, ahol egyetlen egy részletszámítási szabályt sem kell előírni/tudni/(érteni) a sikerhez.



10. ábra: Csökkentett korlátozó feltételszám (forrás: saját ábrázolás)

Összehasonlítás

A klasszikus programtervezés, specifikációkészítés, ill. az ezen belül feltáródó alternatívák mindenkor a konkrét probléma leírásából fakadnak minden részletre kiterjedően.

A Solver-alapú megközelítés tulajdonképpen csak magának a feladatnak az Excel-esítését jelenti, ahol Excel cellákon keresztül kell leírni magát a SUDOKU alaplogikát. Más megfogalmazásban: a Solver-alapúság lényege, hogy bármilyen potenciális megoldás véletlen kiválasztásakor le tudjuk vezetni a hibát. S ennek csökkentésére lehet anélkül bevonni a Solver-t, hogy a Solver tudná, vajon SUDOKU-t old-e meg, vagy bármilyen más feladatot.

A SOLVER tehát olyan quasi univerzális problémamegoldó motorként is felfogható, mely context free jelleggel vethető be. Vagyis a SOLVER kapcsán a problémamegoldást nem kell algoritmizálni soha.

A Solver-es megoldás azonban nem feltétlenül méretfüggetlen, vagyis az Excel licence eleve korlátozza a ténylegesen megoldható feladatok méretét, ami egy célszoftver esetén már csak legfeljebb hardver-korlátokba ütközés esetén értelmezhető probléma.

Következtetések

A fenti gamifikált oktatási elveket is kielégítő alapfeladat és a bemutatott megoldás pár merőben más gondolkodásmódot tükröz vissza. S a maga módján mindkettő tekinthető programozásnak, ahol az Excel macro-k még csak szerephez sem jutnak.

Az általános iskolától az életen át tartó tanulás minden szintjén szükséges lenne a Solver-alapúságra nagyobb hangsúlyt fektetni, különösen akkor, ha a KNUTH-i elvet szem előtt tartjuk: tudás az, ami forráskódba átírható – minden más emberi aktivitás művészet.