A hamis remény kódjai, avagy mi különbözteti meg a jó és a középszerű programozókat egymástól?

(Codes of the false hope – or what are the differences between the good and the norm-like programmers?)

Pitlik László, Pitlik Marcell, Pitlik László (jun), Pitlik Mátyás, Rikk János (MY-X team)

Kivonat: A cikk célja tételes példákkal annak demonstrálása, mi a különbség az adott pillanatban, azaz szűkített minőségbiztosítás mellett megoldásnak sejlő algoritmus(terv) és egy tényleges (univerzális) megoldás között. Az adott feltételrendszer esetén alternatívnak tekinthető, hatásos megoldások kapcsán felmerül az Occam-borotvája elv, vagyis az a kérdés: melyik megoldási alternatíva tekinthető a legjobbnak, azaz legegyszerűbbnek. A programozás(tanulás) többféle stratégiát követhet: lehet ún. logikai blokkokból, más szóval pl. függvény-orientáltan építkezni, s lehet quasi csak a mindenkori programnyelv alapelemeit (ciklust, elágazást, alapműveleteket, stb.) használni. Mivel ezek az alapelemek szinte univerzálisak, így egy programnyelv ismeretében majdnem minden további programnyelv megérhető illik, hogy legyen szinte bárki számára.

Kulcsszavak: pszeudo-kód, rendszermodellezés, rendszertervezés, szoftvertesztelés, MOOC

Abstract: The paper presents examples in order to demonstrate what kind of differences can be identified between a seemingly/temporarily correct and a universal robust solution. Concerning the effective solutions, it can be spoken about the Occam’s razor – what means the most simple solution should be evaluated as the best solution. The executing/learning of programming can follow more strategies: it is possible to use logical blocks (with other words functions) or it is also possible to use only the basic elements of the given language (like loops, conditional branches, basic operations, etc.). These basic elements are in the most languages given therefore it is possible almost each language to learn effectively based on the knowledge about one single language.

Keywords: pseudocode, system-modelling, system-planning, software-testing, MOOC

# Bevezetés

Előzmények:

* <https://miau.my-x.hu/miau/272/anonimtas_az_uj_penz.docx>
* <https://miau.my-x.hu/bprof/autodidakta_kritikus_tanulas_mintazatai_p2.docx>
* <https://miau.my-x.hu/bprof/autodidakta_kritikus_tanulas_mintazatai.docx>
* <https://miau.my-x.hu/bprof/hogyan_jussunk_el_egy_pszeudokodig.docx>, …

Részletek: <https://miau.my-x.hu/miau/272/excel_alapok_l.xls>, ill. <https://miau.my-x.hu/miau/272/excel_alapok_l_alt_pont.xls>

Válasszunk egy látszólag triviálisan egyszerű feladatot és ott és azzal a nyersadat-mennyiséggel oldjuk meg a feladatot Excel-ben. Az, hogy az Excel ki mennyire akarja, meri, szeretné pszeudokódnak, sőt, akár egy fajta programozásnak (vö. Logiscool – block-alapú, ill. mix-alapú kód-írás előkészítő módszertan), nem befolyásolja a lényeget: mindennemű megoldáshoz, így az Excel-alapúhoz is logikus, algoritmikus gondolkodás szükséges.

# A feladat

A feladat legyen egy fajta ECDL (alapozó), azaz gondolkodás-módszertanilag gyorsan átlátható, Excel-technológiailag tetszőlegesen komplex módon kezelhető kihívás: keressük az 1. és a 2. ábra egyetlen egy (zöld) cellában különböző nyers/alap-adatai alapján a legjobb (azaz legmagasabb kerekített) jegyátlaggal rendelkező diáko(ka)t?





1. Ábra: Az 1-elemű megoldás-halmaz esete (forrás: saját ábrázolás)





1. Ábra: A több-elemű megoldás-halmaz esete (forrás: saját ábrázolás)

## Megoldás – 3 egymásba ágyazott függvény 1-elemű megoldáshalmaz esetén

B23=INDEX(A2:A12;HOL.VAN(MAX(H2:H12);H2:H12);1)

A megoldás központi eleme a MAX()-függvény, mely megadja a benchmarkot, vagyis a keresendő aktuálisan legnagyobb értéket. Már itt fel kell, hogy merüljön a programozóban a kérdés: hány főre lesz igaz, hogy holtversenyben a legjobbak? Aki erre nem gondol, az nem gondol arra, hogy minden változó, vagyis a feltett kérdésre adható válaszok halmaza kapcsán is ismerni kell a pontos definíció, inkl. a mértékegységet, s a teljes értelmezési intervallumot. Egy olyan képzésen, mint a BPROF esetén, ahol csak jeles osztályzat van, mert addig kérdeznek és segítenek a konduktorok, míg járni nem tud a páciens, s nem hagyják ott felborulva és a teljesítményété lefitymálva félúton, ott minden diák lehet a legjobb. Ez pedig azt a kérdést is fel kell, hogy vesse, hogy adott változó képes lesz-e tárolni egyáltalán, mint köztes tároló korlátlan nevet tetszőlegesen sok karakterszámmal nevenként és mindösszesen?

A holtverseny léte matematikailag, a holtverseny létéből fakadó méretprobléma pedig rendszerszinten nem kerülhető meg, ahol rendszerszintű kérdéssé válik azonnal, hogy adott céges arculati kézikönyv mellett, mely egy mobil-alkalmazás kapcsán a legkisebb még alkalmazható betűméretet és típust is előírja, mennyi érdemi tartalom jeleníthető meg adott képernyőkép keretében, s ha előreláthatóan nem garantálható, hogy mindenkor minden, akkor miként kell gondoskodni a „lapozásról”? (pl. önálló listanézettel, mely görgethető, x elemenkénti lapozó-navigációval, s ha igen, akkor milyen ugrások legyenek megengedettek a lapozáskor: csak egy-egy előre/hátra lépés x darabonként, vagy lehessen gyorsítva is pörgetni a listát, ill. lehessen-e a végére/elejére ugrani, vagy adott oldalra, vagy adott betűhöz, számhoz…

Az INDEX() és a HOL.VAN() függvények pedig olyan burkok (blokkok), melyek a nyers adatok közötti összefüggések részleteit, ill. láncreakció-szerűen komplexitását képesek kezelni. De vajon szükséges-e 3 egymásba ágyazott (vö. blokk-alapú) függvény-kapcsolat a feladat megoldásához? Nem lehetne-e kevesebb függvénnyel megoldani a feladatot?

## Megoldás – 2 egymásba ágyazott függvény 1-elemű megoldáshalmaz esetén

C23=FKERES(MAX(H2:H12);$H$2:$J$12;3;1)

Az FKERES() kapcsán már elöljáróban el kell mondani, hogy ez a függvény sajnos nem rendelkezik a visszatérési érték helyén a negatív számok használatának engedélyével. Pedig, ha lehetséges lenne a -1, -2, …, -n értékadás is, akkor a jelenlegi szintaktika mellett nem kellene a mindenkor balszélső oszlopként használt tartalom mögé hivatkozni a tőle balra álló tartalmakat, mint jelen esetben a neveket (vö. 1. és 10. oszlop – ill. A-oszlop és J oszlop – vagyis J1=A1).

Ha az FKERES() 3. paraméterének értelmezési intervalluma flexibilisebb lenne, akkor is szükség lenne a keresési érték (1. paraméter helyén) a MAX() függvényre – amíg ki nem váltjuk ezt valamilyen trükkel, ill. függvénytelenítéssel (vö. MAX()/MIN(), sőt FKERES() kiváltása elemi Excel műveletekkel).

## Megoldás – 1 függvény 1-elemű megoldáshalmaz esetén

C23=FKERES(6;$H$2:$J$12;3;1)

A bátrak és/vagy a felelőtlenek megkísérelhetik kiaknázni azt, hogy az FKERES kapcsán létezik az a használati utasításréteg, miszerint az utolsó (4.) paraméter értéke a HAMIS (0), az a keresett kifejezést egzakt módon megtalálni akaró üzemmód mellett egy fajta IGAZ (1), azaz közelítő üzemmód formájában is létezik. Ilyenkor az egzakt módon meg nem talált keresési értékhez legközelebb álló érték kerül értelmezésre, de ez a közelség nem akármilyen módon, hanem a sorba rendezett balszélső oszlop logikáját követve a keresett értékhez elvileg legközelebbi cella által értelmeződik.



1. Ábra: A sorrend-sérülés hatása az FKERES() függvény működésére (forrás: saját ábrázolás)

Vagyis, ha a keresési érték a jegyek 1;2;3;4;5-ös halmazához képest 6, akkor bármi legyen is a valós jegyek maximális értéke egy nyersadattömegben, a 6-hoz mindenkor ez a maximum fog a legközelebb állni szabályos sorrendezés esetén. Ha azonban a sorrend nem zavartalan, akkor a visszatérési érték bármi lehet.

A kíváncsiak számára érdemes végig próbálni a C23 cellát illetően az alábbi paramétereket a 6-os helyén (vö. C23=FKERES(6;$H$2:$J$12;3;1), figyelve, mi lesz ezek hatása: 0;1;1.1;1.7;1.9;2

Ez a kis gyakorló feladat eredmények nélkül is arra hívja fel a figyelmet, hogy a függvényekre és ezek egymásba ágyazására alapozó programozási stratégia csak akkor vezet univerzálisan hibátlan (hatásos) megoldásokhoz, ha a függvények működésével, értelmezési intervallumaival, az összes bemeneti variáns és a kapcsolódó kimenetek minden részletével tételesen tisztában van a programozó. Az, hogy ki van tisztában és mivel, nem is olyan egyszerűen eldönthető kérdés (vö. tesztelés), mert pontszerű/véletlenszerű/szúrópróbaszerű működéshelyesség-ellenőrzések sajnos nem garantálják mindenkor, hogy minden esetben helyes lesz a programban. A tesztesetek tervezésével, a véletlenszerűség csökkentésével, a szélsőségek, specialitások letapogatásával a teljes kombinatorikai tér átellenőrzésétől elvileg adott esetben el lehet tekinteni. De a teljesség ellenőrzése nélkül sosem lehetünk biztosak abban, hogy nem maradt-e valahol egy speciális HA()-feltétel, mely csak pontszerűen okoz zavarokat[[1]](#footnote-1).

## Megoldás – 0 függvény 1-elemű megoldáshalmaz esetén

C23=L13

A 3-2-1-függvényre alapozó megoldások az Occam-borotvája kapcsán (ennek is az alkalmazott függvények számát firtató attribútumát tekintve, mely értelemszerűen a minél kevesebb, annál jobb elv mentén rangsorolandó a későbbi anti-diszkriminatív feldolgozás megalapozásaként) elvileg egyre egyszerűbb és egyszerűbb (azaz ideálisabb) megoldások felé terelik az objektumokat.

A 0 függvény felhasználásának lehetősége egyben átvezet a függvénytelen, csak az adott keretrendszer (programnyelv) alapvető elemeit felhasználni akaró programozási stratégia irányába.

Az 1. és a 2. ábrapár szürkével jelzett részei alapján látható, miként lehet az FKERES()-t is kiváltani: vagyis:

* ha a MAX() függvényt úm. előtétként használva (vö. G13) már tudjuk a valós maximumértéket (vö. későbbi alfejezet a MAX()-függvény kiváltásáról),
* akkor minden egyes sor esetén eldönthető a K-oszlopban celláról cellára, vajon egyenlő-e az adott érték a maximummal, vagy sem
* így, ha csak egy legjobb tanuló van, akkor a K-oszlopban 1 db 1-es jelzés szerepel és minden egyéb cellában 0-s jelzés
* (nem mellesleg ez a megközelítés egyben átvezet a több legjobb esetének, vagyis a holtversenyek létének kezeléséhez)
* A K-oszlop vezérlőjelei alapján az L-oszlopban már csak azok a nevek jelennek meg, melyek esetén a K-oszlop jelezte, hogy ott egy legjobb tanuló van
* (a K-oszlop eredménye egyből is lehetne az L-oszlop tartalma, hiszen az elemi összefüggések is egymásba ágyazhatók, de a K-oszlop léte segít a holtversenyek darabszámát is tisztázni, ami számos későbbi, kiegészítő funkcionalitás alapja: de minimum az önellenőrzésé: pl. hány szóköz van a végső megoldásban, ahol ez a szám+1 azonos kell, hogy legyen a K-oszlop összegével, stb.)
* az L13 cella már csak a felette lévő L-oszlop összefűzését jelenti, s ennek eredménye maga a legjobb tanuló neve (sőt, a legjobb tanulók neve – amíg ezeket nem kell pl. ABC-sorrendbe, vagy a kerekítés mögötti valós teljesítmények sorrendje szerint megadni)…

## Megoldás – 0 függvény több-elemű megoldáshalmaz esetén

C23=L13

A 4. megoldás tekinthető akár az 5. megoldás leírásának is: hiszen formálisan a legjobb tanulók meghatározását maga az FKERES() fentebb leírt kiváltása garantálta.

Itt is meg kell jegyezni, hogy a MAX()-függvény nem integrálása a végső megoldásba, lényegében csalás, vagyis a 4. és 5. megoldás maga még dolgozik a MAX()-függvénnyel (vö. függvények kiváltása fejezet).

## Megoldás – HA()-alapú elágazás

C23=HA(K13<>1;”még nincs megfelelő megoldás”;”erre az esetre már van megoldás”)

Amennyiben valaki legalább már addig eljut, hogy a probléma egyes részeire, pl. a holtverseny hiányára tudatosan mer megoldást kínálni, s a többi esetre még nem, akkor ez egy HA()-függvénnyel jelezhető.

„Ideális” esetben ez a HA()-függvény a megoldás létezik ágon magánan a megoldásnak a képletét tartalmazza. Az idézőjel az ideális eset körül azért indokolt, mert ha ez nem egyetlen egy képlet a HA()-feltétel megfelelő ágán az egyetlen (a részhalmazokat szétválogató) HA()-képletben, akkor ez a HA()-burok minden kapcsolódó helyen szükséges lehet, ami az Occam-borotvája elv kapcsán egy másik attribútum esetén rossz pontokat fog jelenteni, s ez a másik attribútum a forráskódot jelentő karakterek száma (mely minél kisebb, annál jobb).

A HAHIBA() függvény az Excel-ben egy olyan speciális támogatás, mely a normál programozásban a kivételkezelést és/vagy a hibakezelést jelenti. A HAHIBA() függvény kapcsán a holtversenyt jelentő ágat egy hibás kifejezésbe terelve tudatosan, a HAHIBA() függvény is képes adni egy üzenetet arról, hogy ez az ág még nincs kész…

Tudni érdemes, hogy minden olyan esetben, ahol a hibakezelés később az ügyfél felé is kell, hogy jelezzen, ott az hibakezelésnek ergonómiai, hermeneutikai szempontból mesterműnek illik lennie, vagyis az ügyfélnek a kapott hibajelzésen keresztül magát a hibát is el kell tudnia hárítani minden olyan esetben, ahol az ügyfél ehhez megfelelő jogosultságokkal rendelkezik. A hibakódok vissza adás az ügyfél felé egy fajta programozási fasizmus, ami csak az übermensch-ek számára engedi meg, hogy az egyébként detektált és akár robotizálhatóan is elhárítható zavar feloldását az érintett munkakör sikerélményeként is meg lehessen élni (ismét csak megfelelő jogosultságok esetén).

Az egy önálló kérdés, vajon IT-biztonsági szempontból az ügyfél (és/vagy a betörő) miről illik, hogy információt kapjon, ahol a hiba sok esetben már a védelmi rendszer része…

## Megoldás - …

Minden további megoldás feltárása az Olvasó feladata annak érdekében, hogy lehetőség szerint az Occam-borotvája fejezetben bemutatandó lépéseket minden Olvasó saját maga is egyedi adatvagyonon tudja elvégezni – ami önmagában már majdnem az önálló tanulás és önellenőrzés elvárásait, vagyis a MOOC-őssejt elvárásait kielégítő megközelítés egyetlen egy szöveges dokumentummal és ennek XLS-mellékleteivel. De sajnos az egyedi Occam-futtatások ellenőrzése egy speciális értelmező robotot kíván még meg, hasonlóan ahhoz a kihíváshoz, hogy forráskód-hatásosságot robot garantáljon tanári beavatkozás helyett, ill. brute-force vagy szúrópróbaszerű tesztek készítése helyett…

# Függvények kiváltása „elemi” lépéssorokkal és macro-val

Az alábbi fejezet szükségszerűségét a fenti gondolatok már több ponton is megalapozták:

## MAX()/MIN()

A maximális lista érték és/vagy a minimális listaérték megállapítása szinte minden programnyelven beépített függvény, de ha valahol ez még sem lenne adott, akkor muszáj valamit kezdeni a kihívással:

### Macro-nélkül

A MAX() és a MIN() függvény kiváltásának lehetséges lépései közül a 4. ábra bemutatja azt, ahol a lényeg semmi más, mint a mindenkori szélsőérték felismerését, tárolását garantálni, hogy a következő listaértéket már ehhez lehessen viszonyítani:

* A párösszehasonlításkor az inicializálás (sárga cellák) önálló lépésként értelmezhetők.
* Majd a HA() függvény kisebb/nagyobb relációjától függően a celláról cellára történő összevetés és az eredmények kiírása már képletmásolással történik.

Itt érdemes megjegyezni: az Excel nem tud ciklusokban gondolkodni látszólag, de a cellamásolás, ill. az eseteket jellemző cellatartalmak maguk egy fajta ciklusvezérlés-szerűséget jelentenek. A klasszikus programozás kapcsán ez például egy jelentős gondolati átállási kihívás lehet adott személyek esetén adott ideig. De ez cserében garantálja a kód méretfüggetlenségét, amiről eddig nem is volt még szó…

 

1. Ábra: A MAX() és a MIN() kiváltása (forrás: saját ábrázolás)

A kód méretfüggetlensége azt jelenti ugyanis, hogy a nyersadatok sorszámának és tartalmának változása esetén is biztosan a helyes eredményt kapjuk. Az Excelben jelenleg csak az azonos méretű és eltérő tartalmú nyers jelek feldolgozása lehetséges könnyedén. Áthidaló megoldásként a bemeneti jeltartományt lehet óriásira választani, de ebben az esetben fel kell készülni a hiányzó adatok eseteire, összefüggéseire, következményeire, melyre az Excel maga is fel van készítve részletesen (pl. vö. átlag-számítás lyukas listák esetén, ahol az Excel maga határozza meg a lyukas lista hosszához képest a valós adattartalmat jelentő cellák számát, s csak ezen (számértékként értelmezhető) cellákra számít összeget, majd a darabszám alapján átlagot. A HAHIBA() mellett ez is egy olyan Excel-specialitás, melyért nulláról írt forráskód esetén a kezdő programozónak sokat kellhet szenvednie…

### Macro-val, vagy más programnyelven

Ez a pont gyakorló feladatként kerül csak említésre minden Olvasó számára, aki az Excel pszeudokód jellegű megoldását pl. VBA vagy C++, vagy tetszőleges más egyéb keretek között akarja kipróbálni…

## FKERES()

Az FKERES() kiváltását a 4. megoldás mutatta be (függetlenül attól, hogy a MAX() függvény létezik előtét-lépésként, vagy az is kiváltásra kerül a fentiek értelmében).

### Macro-nélkül

Ez a pont gyakorló feladatként kerül csak említésre minden Olvasó számára, aki az Excel pszeudokód jellegű megoldását pl. VBA vagy C++, vagy tetszőleges más egyéb keretek között akarja kipróbálni…

# Occam borotvája

Ez a fejezet egy teljesen új nézőpontot alapoz meg a programozással kapcsolatban. Eddig szó volt a blokk-alapú, függvény-orientált, (ha úgy jobban tetszik egy fajta objektum-orientált) gondolkodásmódokról és ezek ellenpólusaként a minden feladatot programozási alapelemekből felépítő stratégiáról – hiszen minden függvény, minden valamilyen inputokat kezelni képes és valamilyen outputokat produkáló „objektum” ezen alapelemekből épül fel.

A hatékonyság záloga manapság a sok-sok függvényt ismerő, kombinálni tudó, (ill. korábbi kódrészleteket integráló, testre szabó) eljárás - programozás címén. Az ősi és mindenkor univerzális megoldás pedig az elemi eszközkészletre alapozó megoldások keresése lesz és marad. Ha valaki nem csoda-programozó, akkor ne akarja pl. az FKERES()-t, a kimutatásvarázslási megoldásokat kiváltani saját macro-kkal, mert azok futásideje NAGYSÁGRENDEKKEL is rosszabb lehet. Tudni kell azonban, hogy nem minden feladat oldható meg egyedi (elemi eszközöket használni kényszerülő) kódok írása nélkül.

Bár szinte minden programozási nyelv alaplogikája hasonló (lévén a számítógép, ami megszólítanak quasi azonos), mégis van gazdasági/hatékonysági (tudásmenedzsment) szempontból értelme a specializációnak, vagyis egy/több, de nem minden programnyelv nagyon mély kiismerésének.

Az objektum-orientáltság elemzési típusú értelmezése szerint:

* objektumok (jelen esetben alternatív megoldások – melyek rendelkeznek azonos hatásossággal – pl. az 1 győztes kiválasztani tudásával)
* attribútumaik mentén leírhatók
* értékekkel (mérhető, objektíven megállapítható értékekkel)
* egy OAM-ben (objektum-attribútum-mátrixban)…

Vagyis az Occam borotvája elv egy OAM-ot igényel kiindulásként, ahol ebben a cikkben 6 darab számozott alternatív megoldásból 4 darab az, ami deklaráltan a csak egy győztes van esetről szól, de ezek is rendelkeznek variánsokkal, ha pl. a MAX() alkalmazására vagy kiváltására gondolunk. A lehetséges objektumok száma vélelmezhetően végtelen. Így itt és most csak az Occam-borotvája context free és automatizálható elv bemutatásának jelképes méretű demonstrációja történik meg.

Fontos kiemelni: Az Occam borotvája elv azért automatizálható, mert elvileg az algoritmus fogalmához előre bárki képes az összes számára értelmezhető mérőszámot (attribútumot) katalogizálni és automatizált mérési eljáráshoz kötni, hiszen az az attribútum, ami nem automatikusan, objektíven mérődik, nem is attribútum ebben a kontextusban.

Az Occam borotvája elv kapcsán azt is fontos rögzíteni, hogy ez az elv csak onnantól nyer valódi tudományos értelmet, amikortól nem egy-skálás az egyszerűség megállapítása: pl. egy-skálás megoldás az, ahol pl. adott feladat kapcsán a futásidőt mérjük minden egyes alternatív (szoftveres) megoldás esetére azonos hardver-feltételek mellett, azonos adatmennyiségek eseteiben – s minden egyes szcenárió kapcsán igaz, hogy a leggyorsabb megoldás soha nem kaphat más helyezést, csak az első helyezést. Ha bármikor is (pl. valamilyen adatarányok esetén) előáll, hogy egy másik alternatíva a gyorsabb, nem az univerzálisnak remélt győztes, onnantól kezdve a többtényezős evaluáció nélkül nincs értelme a (algoritmus-) egyszerűség absztrakciójáról beszélni, mert a kérdés innentől anti-diszkriminatív jellegű: lehet-e minden azonos hatású algoritmus (alternatíva / objektum) másként egyformán egyszerű (ideális)?

Ennyi bevezetés után íme, az OAM:

Az 5. ábra mutatja be az OAM-ot magát, míg a 6. ábra már a minőségbiztosítás utáni eredményállapotot tartalmazza:



1. ábra: Egy potenciális OAM (forrás: saját ábrázolás)



1. ábra: Occam-borotvája-elv operacionalizálva (forrás: saját ábrázolás)

Az OAM kapcsán tudni kell, hogy

* A használt függvények száma egyszerűen beazonosítható, akár integrálódnak egy/több képletbe, akár előtétként kerülnek meghivatkozásra…
* Az O2 és az O3 egymás antagonistája, mert az előtét logika mindenkor több karaktert használ fel, mint az integrált képlet, így ugyanazon megoldás legalább egy leíró paraméterében (vö. A2: 55>52 karakter) rosszabb…
* Az ismétlődő hivatkozásoknál a cellák növekvő helyiértékeket, azaz szakaszosan egyre több és több karaktert jelentő fizikai megvalósulása nem került figyelembe vételre, csak egy karakterhely a sorszámot illetően az oszlop-azonosító mellé…
* (minden további részlet a háttér-xls-ben az OAM munkalapon látható)

Az eredmények értelmezése:

* A direkt és az inverz nézet, vagyis a preferált irányok alapján (direkt) és ezek ellentettjeként értelmezett 2 OAM eredményei objektum szinten egymás tükörképei, vagyis minden objektum valid…
* Az antagonizmus által érintett 2 objektum közül értelemszerűen az a jobb, mely integráltsága okán rövidebb: vagyis O2>O3
* A győztes a matematikai szempontból leginkább kritikus O4, mely kapcsán a sorba rendezendő nyersadatok sorba rendezési művelet nem kerül büntetésként beszámításra és sorba rendezés nélkül ez a megoldás hamis eredményre vezethet…
* A 2. legjobb az FKERES()-t kiváltani képes elemi megoldás – ami elgondolkodtató – főleg, ha valaki meg is alkotja az FKERES()-t pl. macro formájában…
* Az O1 és az O7 normaszerű, vagyis az alaptananyag (O1) megfelel az elvárásoknak…
* A legrosszabb (legbonyolultabb) objektum az O6, vagyis, ha már egyszer megkezdődik a függvény-mentesítés (vö. O7), akkor ezt érdemes következetesen végig vinni…
* …

# Konklúziók

Az O4 győzelme világosan jelzi, mi is az a csábító erő (vö. hamis remény kódja), ahol a programozó bele mer szeretni valamiféle gyors, egyszerű, a helyzet kapcsán masszív specialitásokat kiaknázó megoldásba, mely univerzalitása olyan korlátozott is lehet, hogy még elemei sem hasznosíthatók az adott feltételek mellett győztes megoldásnak az univerzális megoldásban.

# Referenciák

…szövegközben…

1. Az egyik nyugdíjbiztosításokat közvetítő nemzetközi cég kapcsán derült fény véletlenszerűen arra, hogy adott napon születettek esetén a rendszer biztosítási ajánlatában egy sor téves, s minden más születésnap esetén helyes. Ez felveti természetesen, hogy a tesztelés nem csak kombinatorikai alapon, hanem strukturális alapon is szükséges: hiszen egy ilyen zavart okozó HA()-feltételnek léteznie sem lenne szabad. Ez a gondolat máris tovább vezet a forráskódok robotok általi értelmezéséhez, evaluációjához, vagyis az emberi szakértő robottal való helyettesíthetőségének kérdésköréhez… [↑](#footnote-ref-1)