Anomália mértéke a virtuális lapozott memóriában

Fornai Péter
ELTE TTK nappali tagozat
Programtervező Matematikus

Témavezető: Iványi Antal
ELTE TTK, Budapest, 1997.

EÖTVÖS LORÁND TUDOMÁNYEGYETEM

TERMÉSZETTUDOMÁNYI KAR

INFORMATIKA TANSZÉKCSOPORT

DIPLOMAMUNKA-TÉMA BEJELENTŐ

Név: **Fornai Péter**

Tagozat: **nappali** Szak: **Programtervező Matematikus**

Témavezető neve: **Iványi Antal**

munkahelyének neve és címe: **ELTE TTK, 1088 Múzeum körút 6-8.**

beosztása és iskolai végzettsége: **Egyetemi Tanár**

A dolgozat címe: **Anomália mértéke a virtuális lapozott memóriában**

A dolgozat témája: Lapozott virtuális memóriákban az anomália mértékének a vizsgálata, ezenbelül a FIFO algoritmusnál a korábban a Commuication of the ACM-ben megjelent maximális értékre vonatkozó sejtés megdöntése, illetve a maximum pontos meghatározása, igazolása, illetve több algoritmus kidolgozása a probléma vizsgálatához. Azaz szakdolgozat szintű kidolgozása annak a témának, amellyel jelenleg is foglalkozom a témavezetőmmel.

A témavezetést vállalom:

Iványi Antal
(a témavezető aláírása)

Kérem a diplomamunka témájának jóváhagyását.
Budapest, 1996. október 18.

Fornai Péter
(a hallgató aláírása)

A diplomamunka-témát az Informatikai Tanszékcsoport jóváhagyta.
Budapest, 1996 .....

Benczúr András
dr.Benczúr András
tanszékcsoportvezető egyetemi tanár

Tartalom

A VIZSGÁLT PROBLÉMA RÖVID ISMERTETÉSE 4

A PROBLÉMA IRODALMÁNAK ÉS ELŐZMÉNYEINEK ÁTTEKINTÉSE 5

IRODALMI ADATOK A FIFO ALGORITMUS ALKALMAZÁSÁRÓL 5
A LAPCSERÉLÉSES SZÁMÍTÓGÉPEK, A FIFO ANOMÁLIÁJA 6
ELŐZMÉNYEK, A DOLGOZAT TÉMÁJA 8

A PROBLÉMA MEGOLDÁSÁNAK ISMERTETÉSE 9

MATEMATIKAI MODELL 11

MI AZ ANOMÁLIA? 12

PÉLDA 15

1. PÉLDA 15
2. PÉLDA 18

EGY LAPSOROZAT ELŐÁLLÍTÁSA A KISEBB MEMÓRIÁBAN 20

FELSŐ KORLÁT 23

A FELSŐKORLÁT FINOMÍTÁSA 25

NÉHÁNY SZÓ AZ M PÁROS ESETRŐL 28

SZÁMÍTÓGÉPES SZIMULÁCIÓ EREDMÉNYEI 29

MOLNÁR MIHÁLY ALGORITMUSA 29
KERESŐ ALGORITMUS: MAXIMÁLIS ANOMÁLIA 30

AZ EREDMÉNYEK ÖSSZEFOGLALÁSA 32

PROGRAMOK 33

KERESŐ PROGRAM: MAXIMÁLIS ANOMÁLIA 33
MOLNÁR MIHÁLY PROGRAMJA 36

IRODALOMJEGYZÉK 39

TÁRGYMUTATÓ 40

A vizsgált probléma rövid ismertetése

Bizonyos programok futtatásakor előfordul, hogy annak ellenére, hogy a memória méretét növeltük, a szükséges lapcserék száma mégis növekszik! Ezt a jelenséget hívjuk Béládi anomáliának.

A dolgozat témája ezzel kapcsolatos.

Lapozott virtuális memóriákban az anomália mértékének vizsgálata a FIFO (First In First Out) algoritmus alkalmazása esetén. A [2]-ben megjelent sejtés megdöntése, illetve az elérhető maximum meghatározása. Annak kidolgozása, hogy konkrét paraméterek mellett milyen korlát teljesül az anomália mértékére, és annak kimutatása, hogy ez a korlát már nem finomítható tovább. Illetve néhány a téma kidolgozás közben kifejlesztett szimulációs algoritmus érdekes futási eredményeinek a bemutatása.

A dolgozat a feladatot úgy oldja meg, hogy először konkrét paraméter választások esetén megmutatja, hogy mi az a felső korlát, amely mindenképp fennáll az anomáliára. Miközben bemutatjuk, hogy milyen paraméter választás mellett lehet elérni a legnagyobb anomáliát (az előzőekben meghatározott korlát segítségével), meghatározzuk az anomália pontos maximális értékét.

Az anomália maximuma (helyesebb szuprémumot mondani) csak egy optimális helyzetből indulva, és onnan egy sorozatot iterálva lesz elérhető. Ezért mindjárt a dolgozat elején szerepel egy algoritmus, amely megmutatja, hogy egy előre megadott helyzetet el lehet érni – bizonyos esetektől eltekintve, amelyek azonban nem játszanak szerepet a későbbiekben. A dolgozat ezen kívül tartalmaz még két algoritmust a probléma vizsgálatához.

A dolgozat tartalmaz egy „A probléma megoldásának ismertetése" című fejezetet, amely ismerteti a dolgozat okfejtésének vezérfonalát. Az ott szereplő fogalmakat a további fejezetekben definiáljuk. A problémakörbe mélyebb betekintést „A probléma irodalmának és előzményeinek áttekintése", valamint a használt „Matematikai modell" című fejezetek nyújtanak.

A probléma irodalmának és előzményeinek áttekintése

A bevezetés két részből áll. Az első röviden bemutatja, hogy a vizsgált algoritmus széles alkalmazási területén belül hová kapcsolódik szervesen az általam vizsgált probléma terület. A második rész már a választott terület fejlődését tekinti át.

A témakör átfogó ismertetését Iványi Antal [1] operációs rendszerekről szóló előadásai adhatják.

A szakirodalomban a következő kulcsszavak alapján tájékozódhatunk:

FIFO	First In First Out
FCFS	First Came First Served
Scheduling	Ütemezés
Paging	Lapozás
Replacement algorithm	Lapcseréléses algoritmus
Demand paging	Igény szerinti lapozás
Anomaly	Anomália
Virtual memory	Virtuális memória

A dolgozat irodalmának felkutatásához a matematikus könyvtárban megtalálható Mathematical Reviews CD-s változatát használtam, amely gyorsan és kényelmesen teszi lehetővé a keresést a fenti kulcsszavakkal. A legtöbb cikk megtalálható a BME központi könyvtárában vagy az OMIKK-ban.

Irodalmi adatok a FIFO algoritmus alkalmazásáról

A dolgozat fő témája a FIFO (FIRST IN FIRST OUT) algoritmus, amelynek lényege, hogy a kiszolgálásra váró dolgok egyfajta sorban állnak: a végére lehet csak bekerülni, és csak az elejéről lehet kikerülni.

A FIFO algoritmust – bár nagyon egyszerű – mégis még mindig relatíve nagy iránta a tudományos érdeklődés, például a [10] egy 94-ben megjelent cikk még amely a FIFO algoritmus stabilitásának kérdésével foglalkozik a fogyasztó hálózatok kapcsán, kimutatva, hogy az algoritmus bizonyos esetekben instabil lehet. A sorban állási hálózatok területén gyakran előfordul, hogy a FIFO jelleg már eleve adott a „sorban állás" ténye miatt, és ezen kereteken belül próbálnak jobb vagy optimális vagy valamilyen peremfeltételeket vizsgálni [14].

A fogyasztó hálózatok esetében a FIFO algoritmus elég általánosan használt algoritmus (ld. *queueing* elmélethez kapcsolódó nagy mennyiségű irodalmat).A *queueing* (sorban állás) kapcsán talán helyesebb lenne az FCFS (First Come First Served) elnevezés a FIFO helyett utalva ezzel arra, hogy itt „fogyasztók" érkeznek és állnak sorban. A FIFO kapcsán lehet úgynevezett negatív érkezést is megengedő FIFO alkalmazásokkal foglalkozni [11], azaz a sorba bekerült elemek kikerülhetnek onnan. Erre lehetne példa az, amikor egy program futását megszakítják, miután igényt jelentett be egy nyomtatóra, de a nyomtatás még nem kezdődött meg.

Ha a FIFO alkalmazási területét jobban leszűkítjük a programok ütemezése a számítógépeken, akkor a *scheduling* (ütemezés) területére érkezünk. Például a magyarul megjelent [13] azzal foglalkozik, hogy a FIFO-t is tartalmazó algoritmus osztályra a perifériák kiszolgálásakor a

foglaltsági periódus várható értéke azonos; ez a cikk 1982-ben a több perifériával is rendelkező számítógépeket vizsgálta.

A dolgozat az ütemezés területén belül egy szűkebb FIFO alkalmazást kíván célba venni: a *paging machine* (a lapcseréléses számítógép). Ez a FIFO-nak egy érdekes alkalmazási területe, hiszen itt megengedjük, hogy olyan lapok, amelyek már a FIFO sorban állnak (benn vannak a memóriában) ismét előfordulhassanak, azaz lehessen rájuk újabb hivatkozás.

A lapcseréléses számítógépek, a FIFO anomáliája

Már viszonylag hamar felmerül a programozás történetében az igény, hogy nagyobb tárterületet tudjanak a programozók felhasználni, mint ami ténylegesen rendelkezésre áll fizikai központi tár formájában. A dinamikus tárfoglalás területének fejlődéséről [3] adott számot 1968-ban.

Először a szegmensek fogalma jelenik meg, majd az overlaying technika, amely fogalmak még ma is élnek a számítástechnikában. Felmerült azonban azonban egy olyan módszerre az igény, amely egy időben több felhasználó programjának futtatását jobban lehetővé tenné, és nőtt az igény arra, hogy a memóriagazdálkodás gondját levegyék a programozó válláról, és azt automatizálják.

1965-ben publikálta a MAC, Bell és a General Electric Company egy közös kísérleti programját. Létrehoztak egy rendszert, amely 2 processzort tartalmazott, 128 Kszavas tárral rendelkezett. A mottójuk ez volt: „kicsi, de hasznos." A rendszer célja az volt, hogy nagyszámú felhasználót tudjon a gép egyszerre kiszolgálni. A memóriát egyforma méretű lapokra osztották. Egy program gyakorlatilag tetszőleges számú lapot tartalmazott, de csak bizonyos számú lapot tartott a számítógép egyszerre a memóriában. Ha a program egy olyan lapjára hivatkozott, amely nem volt a belső tárban, akkor az operációs rendszer automatikusan behozta a hivatkozott lapot, egy valamilyen algoritmus szerint kiválasztott másik lap helyére. Ennek a módszernek a neve: *demand paging* (igény szerinti lapozás).

M44 néven [6] létrehoztak egy számítógépet, amelynek célja kettős volt:

1. Tesztelés és kiértékelés.
2. A lapozásos számítógépek teljesítményének mérése.

A gépnek 16K lapozható tára volt, de virtuálisan 2 millió szót tudott kezelni. Eredetileg a FIFO algoritmussal oldották meg a lapozás problémáját, hiszen ez a legegyszerűbb és legtermészetesebb választás. A fő szempont igazából az volt, hogy könnyen lehessen a kezdeti operációs rendszert nyomon követni.

Már ekkor ismert volt, hogy a FIFO algoritmus nagyon rossz választás, mint ahogy az [9]-ben bemutatásra kerül, hogy a nem sokkal bonyolultabb LRU (Least Recently Used) algoritmus sokkal jobb választás nála, és Bélády [4]-ben részletesebben is foglalkozott a lapcseréléses algoritmusokkal, és a FIFO a legrosszabb hatékonyságú algoritmusnak bizonyult. Ennek ellenére a FIFO-t a mai napig használják különböző programokban, amelyeknél szükséges a lapozás vagy valamely lapozással ekvivalens fogalom (pl. némely fordító programnál, web alkalmazásnál: HTML dokumentumok szervere stb.).

[6]-ban a szerzők a FIFO kapcsán rámutatnak egy érdekes hibalehetőségre, bár utalnak arra, hogy ez szerencsére nem okozhat valódi gondot egy futó rendszernél:

1. Egy programrész hivatkozik egy másik lapon szereplő adatra.

2.A program éppen futó része azon a lapon van, amelyet most ki kell vinnie a FIFO-nak, ezzel az az érdekes helyzet áll elő, hogy az adat bent van, de a program futó része nincs.

3.Következésképpen vissza kell hozni a futó részt a memóriába. Ekkor lehet, hogy az ütemezési késés miatt a azokat az adatokat viszi ki a FIFO, amelyeket a 2. pontnál behozott.

Bár ilyen probléma nem igazán fordulhat elő, hiszen a belső tár általában lényegesen több lapot tartalmaz, mint a futó programok száma, és ez elég a patthelyzet feloldásához, azért érdemes utalni rá, hogy az LRU algoritmus választásával a fenti helyzet nem is fordulhatna elő. Azaz már ezzel is jobb az LRU, mint a FIFO.

Az M44-es történelmi szerepe abban áll, hogy segítségével igazolták a lapcserélés módszerének gyakorlati működőképességét.

Fontos még megemlíteni az ugyancsak az M44-esen folytatott kísérletsorozatot [5], amelyben részletesen vizsgálták a felhasználó szempontjából a lapcseréléses memória gazdálkodás előnyeit, hátrányait, illetve felhívják a figyelmet arra, hogy a lapcseréléses technika általános esetekben valóban leveszi a gondot a programozó válláról, de nagy adatok kezelésekor (a cikk nagy mátrixokkal foglalkozik) a programozónak foglalkoznia kell az adatok memóriában való elhelyezkedésével, ha nagyságrendekkel hatékonyabb programot akar.

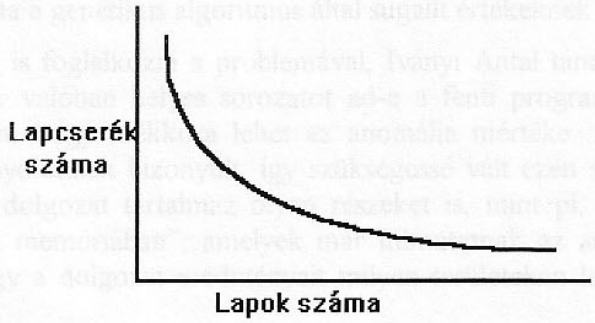
Az időbeni eltérés abból adódik, hogy a lapcsere – azaz amíg a háttértárról behozunk egy lapot a központi tárba – sok időt igényel [1], míg a már bentlévő lapokra már gyorsan lehet hivatkozni.

A fenti különbség nem elhanyagolható. Gyakran a program futási ideje elhanyagolható a lapozásra fordított időhöz képest.

A fenti cikk tesztadatokat mutat arra is, hogy a FIFO lapcseréléses algoritmus általában nem jelentősen rosszabb a MIN algoritmus választásánál. A MIN az optimális, legkevesebb lapcserét igénylő sorrendben kezeli a memóriát. Bár tény marad az, hogy bizonyos algoritmusoknál már jelentős lehet a különbség.

Az 1994-ben megjelent operációs rendszerekről szóló könyv [8] Bélády anomáliaként hivatkozik arra a jelenségre, amelyet ebben a szakdolgozatban részletesebben kívánok elemezni.

Ha azonos méretű lapokat feltételezünk, akkor egy kapcsolatot lehet kimutatni a rendelkezésre álló lapok száma, avagy a rendelkezésre álló lapozásos memória mérete és a programok futatásához szükséges lapok száma között. Ha több lap lehet a memóriában, akkor kevesebb lapozásra van szükség:



Egy jó kérdés az, hogy a lapok száma, azaz a rendelkezésre álló lapozásos memória mérete változhat-e konkrét rendszereknél. A válasz igen, erre példaként szolgálnak azok a programok,

amelyeknél lehet paraméterezni a rendelkezésre álló tár területet. Web-es alkalmazásoknál az az érdekesség, hogy ott a tárterület nem a belső memória, hanem a háttér tár, míg a lassú memória párja a távoli gép.

Egy másik példa lehetne erre a már az M44-es architektúránál is alkalmazott technika, amelynél ki lehetett választani olyan lapokat, amelyek nem vettek részt a lapozásban. Erre lehet példa az operációs rendszer lapjai, vagy például a gyakrabban használt programokat tartalmazó lapok. A FIFO lapozásban csak a virtuális memória többi része venne részt.

Előzmények, a dolgozat témája

Bizonyos programok futtatásakor azonban előfordul, hogy annak ellenére, hogy a memória méretét növeltük, a szükséges lapcserék száma mégis növekszik! Ezt a jelenséget hívjuk Bélády anomáliának.

Sikerült azt megmutatni [2]-ben, hogy ezen anomália jelensége csak bizonyos feltételek fennállása esetén alakulhat ki. A cikk szerzőinek sikerült megmutatniuk, hogy a kettőt tetszőlegesen meg lehet közelíteni, ha az általuk megadott feltételek fennálnak. Ugyanebben a cikkben a szerzők megfogalmaztak egy sejtést is, hogy az anomália mértéke nem érheti el a kettőt, azaz a szükséges lapcserék száma nem nőhet az eredetinek a kétszeresére, ha a memória méretét növeljük.

Ez a dolgozat megmutatja, hogy a kettő nem felső korlát az anomáliára, hanem az anomália tetszőleges határon túl nőhet, ha a memória méretét szabadon megválaszthatjuk. Konkrét memóriaméret mellett a dolgozat megmutatja, hogy mi a tényleges felső korlát, illetve azt, hogy ezt mikor lehet elérni. A dolgozat tartalmaz algoritmusokat az anomália vizsgálatára, amelyek közt van olyan, amely igen érdekes kísérleti eredményt ad.

A problémát Iványi Antal vetette fel, hogy gondolkodjunk el a fenti cikkben közreadott sejtésről. Először Molnár Mihály kezdett foglalkozni a témával és genetikus algoritmusokkal próbálta megoldani, így próbált olyan sorozatot találni, amelynél az anomália értéke elére vagy meghaladja a kettőt. Sajnos azonban hosszabb sorozatokra az algoritmus konvergenciája nagyon rossz volt, így fel kellett ezt a megközelítést adnia.

A végrehajtott kísérletek azonban rámutattak egy érdekes szabályra az anomália kapcsán – erről részletesebben Molnár Mihály algoritmusáról szóló fejezet ír. Konstruált egy programot, amely ezt a szabályt használta, de ettől eltekintve véletlenszerűen választotta a lapokat egy hivatkozási sorozat létrehozásához. A próbálkozás meghozta az eredményt: az anomália értéke kettő fölé emelkedett – bármikor is indította el a programot, annak ellenére, hogy a lapválasztás majdnem teljesen véletlenszerű volt. (Megjegyzés: a program paramétereit megfelelően választotta a genetikus algoritmus által sugallt értékeknek megfelelően.)

Ekkor kezdtem el én is foglalkozni a problémával, Iványi Antal tanár úr kérésére azt kellett megvizsgálnom, hogy valóban helyes sorozatot ad-e a fenti program, és ha igen, akkor el kellett gondolkodnom, hogy mekkora lehet az anomália mértéke. A felvetett problémakör azonban egészen bonyolultnak bizonyult, így szükségessé vált ezen szakdolgozat formájában való kidolgozása. A dolgozat tartalmaz olyan részeket is, mint pl. „tetszőleges részsorozat kialakítása a kisebbik memóriában", amelyek már túlmutatnak az anomália vizsgálatán, bár annak a kérdése, hogy a dolgozat eredményeit milyen területeken lehet hasznosítani, még a jövő kérdése.

A probléma megoldásának ismertetése

A „Matematikai modell" fejezete ismerteti az alkalmazott jelölésrendszert, pontosan definiálja a FIFO és az anomália fogalmát. Leírja, hogy hányféleképp érhetjük el az anomália szélső értékét és azt, hogy ezekre hogyan lesz hivatkozás később. Az anomália szempontjából három paraméter játszik szerepet: a kis és nagy memória mérete, valamint a program lapjainak száma. Ezen rész tartalmazza, hogy a paraméterek milyen megválasztásával nem kell már foglalkoznunk a későbbiekben. A későbbiek során a dolgozat feltételezi, hogy a paramétereket így választottuk meg.

A „Példa" fejezet tartalmaz két példát arra, hogy valóban meg lehet haladni a korábban [2]-ben közreadott sejtést. Mivel a dolgozatban alkalmazott jelölésrendszert nem egyszerű első olvasásra követni, ezért ez a rész egyben be is mutatja, hogy a következő részek miről fognak formálisan szólni. Bemutatja a kezdeti optimális helyzet elérésének algoritmusát, valamint bemutatja, hogy a „páros és páratlan" eset miben tér el egymástól – ezért a két példa.

A következő fejezet bemutatja, hogy annak ellenére, hogy a nagyobb memóriában csak egy megadott lapsorozatot ismételhetünk, kialakítható a kisebb memóriában tetszőleges lapsorrend. Ez a fejezet egyben utal rá, hogy nem igaz, hogy eközben meg tudjuk mondani, hogy a nagyobb memóriában az éppen iterálás alatt lévő lapok közül melyik következik. Szerepel itt egy lemma arra vonatkozóan, hogy az utoljára valahová behozott lapnak mindkét memóriában szerepelni kell. Ennek kapcsán definiálja a „hivatkozás négy típusát", és utal rá, hogy a IV. típust nem is kell figyelembe vennünk. Egy verbális definíciót ad a „ciklus" fogalmára, amelyet a szöveges magyarázatokhoz használok. Az itt szereplő algoritmus konstruktív bizonyítása a fenti tételnek. Helyesség és teljesség bizonyításához szükséges elő- és utófeltételeket megadtam hozzá.

A „Felső korlát" című fejezet ad egy bármely paraméter választás esetén érvényes felső korlátot. A későbbiek alapján látható lesz majd, hogy megfelelő paraméter választás mellett ez a korlát már csak nagyon kis mértékben javítható. Ez az a korlát, amelyet a dolgozat arra használ, hogy megmutassa, hogy más-más paraméter választás mellett az anomália nem fogja elérni a legjobb paraméter választás mellettit. Ebben a fejezetben kikötöm, hogy a továbbiakban a program lapjainak számát a nagyobb memória mérete alapján korlátozom, mivel az anomália szélső értéke szempontjából ez nem játszik szerepet.

„A felső korlát finomítása" című fejezet ismerteti az anomália szélső értékére vonatkozó tételt, amelynek igazolása a fejezet és a dolgozat témája.

Az a.) részben azt látom be, hogy az előző fejezetben leírt általános érvényű korlátot nem is lehet elérni, mivel vannak olyan plusz lapcserék, amelyek azt lerontják.

A b.) részben azt látom be, hogy a tételben megadott korlát elérhető, ha a kisebb memória méretét páratlannak választom, és természetesen a többi paramétert is úgy választom meg, hogy a korlát elérhető legyen. Itt definiálom is egyben formálisan az ezt elérő hivatkozási sorozatot, amely megegyezik a korábban leírt első példában lévővel, ha a paramétereket ugyanannak választom. Az itt szereplő hivatkozási sorozat csak egy optimális helyzetből indítva produkálja a maximális anomália értéket. Az optimális helyzetet a korábban megadott algoritmussal el kell érni az itt leírt módon. Ekkor azonban az anomáliáról már csak szuprémumként lehet beszélni, itt szerepel, hogy mihez fog tartani az anomália, de a pontos érték kiszámítása csak a d.) pontban szerepel.

A c.) rész azt mutatja meg, hogy más paraméter választás mellett, azaz ha az értékek nincsenek szorosan egymás mellett, akkor már nem érhetnénk el a tételben megadott anomália értéket.

A d.) pontban a pontos érték meghatározása szerepel, mivel korábban b.) pontban kihozott értékről csak optimális helyzetből indulva lehet beszélni. Ráadásul kiderül, hogy bár ott egy egész jó tulajdonságú cseresorozatot határoztunk meg, azt csak akkor lehet felhasználni, ha a kicsi memória mérete páratlan. Az I. rész ekkor meghatározza a pontos értéket. A II. részre azért van szükség, hogy megmutassam, hogy páros esetben az I. pontban kiszámított érték nem érhető el.

Itt kell megjegyeznem, hogy bár a II. pontban megmutattam, hogy a páros eset rosszabb, mint a páratlan. A páratlan eset, mint felső korlát érvényes rá, de a pontos értékét nem sikerült bizonyítani. A „Néhány szó a [...] páros esetről" alcím alatt ismertetem a pontos értékre vonatkozó sejtésemet. Leírom, hogy, mi okozza a bizonyítás nehézségét, illetve adok egy olyan hivatkozási sorozat definíciót, amely eléri ezt az értéket. Ez a definíció megegyezik a korábban leírt első példában lévővel, ha a paramétereket ugyanannak választom. Itt továbbá utalok arra, hogy az ebben a részben ismertetett módszer a hivatkozási sorozatra általánosan használható olyan anomália eléréséhez, amely közel áll a maximálishoz.

A „Számítógépes szimulációs eredmények" fejezetben két algoritmust ismertetek. Az első Molnár Mihály algoritmusa, amely annak felfedezéséhez vezetett, hogy anomália értéke nem korlátos, ha a paraméterek tetszőlegesek. A második algoritmus az előző tétel páros esetéhez kapcsolódik, ehhez szükséges vizsgálatokat végeztem vele. Mindkét algoritmusnak több változata létezik. Itt csak a három legfontosabb szerepel. Ezen algoritmusok csak szimulációs szempontból játszanak szerepet, így semmilyen bizonyítást nem szerepel velük kapcsolatban.

Az „Eredmények összefoglalása" fejezet tartalmazza a dolgozat fontosabb eredményeit.

Végül, tényleg csak a teljesség kedvéért – mivel a dolgozat célja nem egy program volt – a dolgozat végén szerepel a fenti algoritmusok egy lehetséges programja, amelyek megírására még a matematikai formalizmus megszületése előtt került sor kísérleti céllal.

Matematikai modell

A dolgozat jelölésrendszer igyekszik követni a [2]-ben leírtakat.

Olyan számítógépet vizsgálunk, ahol a program azonos hosszúságú részekre, lapokra van osztva. Ezek száma N. Ezek halmaza a következő: $A = \{ a_1, a_2 ..., a_N \}$. Gyakran az $1,2,...N$ számokkal azonosítom őket.

A program végrehajtását az $S=(r_1, r_2, ... r_t)$ hivatkozási sorozattal modellezzük, ahol t a sorozat hossza (nem feltétlenül véges).

Érdemes észrevenni, hogy a fenti jelölésrendszer nem csak egyetlen programra értelmezhető. Tekinthetjük úgy is, hogy az A halmaz a virtuális memória lapjait jelöli, és az S sorozat pedig az architektúrából vagy az operációs rendszer választásából adódó lapsorozat. Azaz azt jelöli, hogy a virtuális memóriának éppen melyik része „van benn" a központi tárban. Ez annyit jelent, hogy az itt ismertetésre kerülő modell használható multiprogramozott számítógépek jellemzésére is.

Definiáljuk a következő függvényeket:

„fej" operátor:

$h: A^* \times N \rightarrow A^*$

$$h(s_1 s_2 ... s_q, k) = \begin{cases} s_1 s_2 ... s_q & \text{ha } q<k, \\ s_1 s_2 ... s_k & \text{ha } q>=k. \end{cases}$$

A fej operátor jelentése, hogy egy sorozat – a hivatkozási sorozat – első k tagját veszi.

„vége" operátor:

$t: A^* \times N \rightarrow A^*$

$$t(s_1 s_2 ... s_q, k) = \begin{cases} s_1 s_2 ... s_q & \text{ha } q<k, \\ s_{q-k+1} ... s_k & \text{ha } q>=k. \end{cases}$$

A vége operátor jelentése, hogy megmondja, hogy mi volt az utolsó k lap, amelyre a vezérlés hivatkozott az adott sorozatban.

A lapbevitel jelentése az, hogy ha a memóriában még nem szerepelt az aktuálisan hivatkozott lap, azaz amelynek tartalmára most lenne szükség, akkor azt be kell hozni a memóriába. Ez azzal jár, hogyha a memória tele van, akkor valamit ki kell onnan vinni. Azaz szükség van valamilyen választási stratégiára, amely megmondja, hogy melyik lapot vigyük ki a memóriából. A mi esetünkben ez a stratégia a FIFO (First In First Out), azaz a lap, amelyet legrégebben hoztunk be a memóriába, az fog most kikerülni onnan.

Ennek leírása a következő:

Jelentse $r_j \in S$ azt, hogy r_j hivatkozás szerepel S-ben. $\#S$ jelentse az S-ben lévő hivatkozások számát.

A f szimbolizálja a *lapbevitel* sorozatot:

$f: A^* \times I \rightarrow A^*$

$f: A^* \times N \rightarrow A^*$

$f(\text{üres}, k) = \text{üres}.$

$$f(R\, a_j, k) = \begin{cases} f(R,k) & \text{ha } a_j \in t(f(R,k),k), \\ f(R,k)\, a_j & \text{különben.} \end{cases}$$

ahol: $a_j \in A$, a most hivatkozott lap, és k a memória mérete lapokban.

A fenti jelentése, ha futás közben már végrehajtottuk az R hivatkozási sorozatot, és most a_j következik, akkor két lehetőség van:

1. Ha már bent van a hivatkozott lap a memóriában, akkor nem kell csinálni semmit.
2. Ha nincs bent, akkor ezt most be kell hozni, azaz bekerül a *lapbevitel* sorozatba.

A FIFO jelleg miatt a *lapbevitel* sorozat utolsó k darab tagja megadja, hogy milyen lapok tartózkodnak a memóriában.

Vezessük be a következő egyszerűsítő jelölést:

$R_i := h(R, i)$.

Ennek jelentése az lesz, hogy az i. laphivatkozás után eddig milyen lapok szerepeltek a hivatkozási listán.

Az anomália vizsgálatához meg kell különböztetnünk egy „kicsi" és egy „nagy" méretű memóriát. Ezeken végrehajtva ugyanazt a hivatkozási sorozatot összehasonítjuk a szükséges lapcserék számát.

Legyen adva a kis és nagy memória mérete a következő számokkal lapokban:

m, M, $m<M$.

Vezessünk be további egyszerűsítő jelölést:

$\underline{m}_k := t(f(R_k, m), m)$.

$\underline{M}_k := t(f(R_k, M), M)$.

Ez a k-adik hivatkozás feldolgozása után jelöli, hogy mi található a kicsi, illetve a nagy memóriákban.

$\#\underline{m}_k$ és a $\#\underline{M}_k$ jelentse a lapbevitel sorozat hosszát, azaz a szükséges cserék számát a k-adik lépésben.

Eddig a matematikai modell.

Még használni szeretném a későbbiekben a következő egyszerűsítő jelölést: legyen $\underline{M}$ és $\underline{m}$ a memóriák „aktuális" állapota. Például az i. időpillanatban – az i. lapcsere után – $\underline{M} := t(\underline{M}_i, k)$ azzal a kitétellel, hogy csak $k>i$ lépés után értelmes a művelet, de nem is fogom más esetekre használni.

Mi az anomália?

Az anomáliának azt a jelenséget nevezzük, amikor egy adott S (k hosszú) hivatkozási sorozat feldolgozása után a kisebb méretű memóriában ($\underline{m}$) kevesebb lapcsere történik, mint a nagyobban ($\underline{M}$) –ez pedig ellentmond a várt értékeknek.

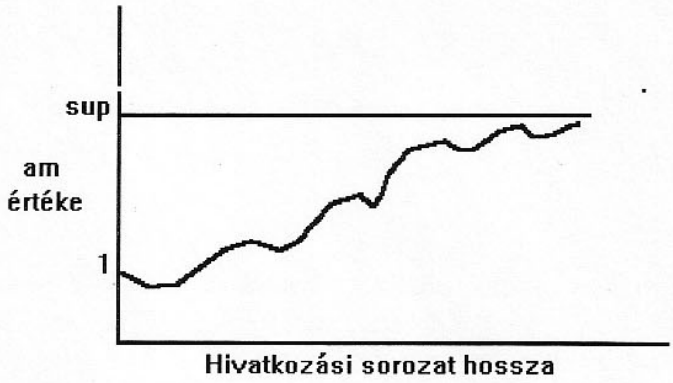
Az *anomália mértékét* a következő hányadossal jellemezhetjük:

$am := (\#\underline{M}k)/(\#\underline{m}k)$.

Akkor beszélünk anomáliáról, ha $am>1$, azaz a kis memóriában ugyanazon hivatkozási sorozat végrehajtása esetén kevesebb lapcsere történt, mint a nagyban.

Két esetet lehet megkülönböztetni: az egyik, ha létezik egy maximális érték valamilyen véges hivatkozási sorozathoz. Ilyenkor az am, az anomália mértéke valahogy váltakozik, de a teljes sorozat végére eléri a maximumot, és ha folytatnánk, akkor biztos, hogy csökkenne az érték:

A másik, ha nem, azaz csak tetszőlegesen megközelíthető értékről beszélhetünk:



Látni fogjuk, hogy a mi esetünkben ez a második lehetőség áll fenn. Ez azt jelenti, hogy minnél hoszabb sorozatokat választunk, annál jobban megtudjuk ezt a korlátot közelíteni. Ezt úgy lehet elérni, hogy először – bármilyen áron – elérünk egy jó helyzetet, majd innen ugyanazt a részsorozatot ismételgetve egyre közelebb kerülünk a szuprémumhoz. Azt nem érhetjük majd el, mivel a jó, optimális helyzetet mindenképp ki kell alakítani, és a mi részsorozatunknál nincs jobb választás. Ld. alább:

Ebben az esetben az *anomália* mértékén a következőt érthetjük:

$$limam := \sup_{S \in A^*} (\#\underline{M}k)/(\#\underline{m}k).$$

Ha ki tudom mutatni, hogy még egy optimális helyzetből indulva is ha igaz, hogy $\underline{M}$-be X lapot hozok be, akkor ez alatt mindenképp Y lapot kellett behoznom $\underline{m}$-be, akkor biztos fennáll a $limam <= X/Y$ felső korlát.

Ha az is igaz, hogy ezután ennek az X, Y db lapnak a behozását ciklikusan tudom biztosítani, és az optimális helyzetet egy véges hivatkozási sorozattal elérem ($K1$ cserével $\underline{M}$-ben, és $K2$-vel $\underline{m}$-ben), akkor:

$limam = sup\ ((K1 + n*X) / (K2 + n*Y))$, mivel $K1/K2$ biztos kisebb X/Y-nál, így:

limam = X/Y.

Megjegyzés: mostantól kezdve a limam-ot fogom használni az anomália jellemzésére, és külön ki fogok térni arra, hogy véges sorozattal ez nem elérhető, azaz a fenti módon lehet kezelni az anomáliát.

A **ciklus** fogalmát (saját szóhasználatom) a következő fejezetben a példa kapcsán mutatom meg, és az ott használt algoritmus leírásánál használom.

Esetekre bontás:

A [2]-ben bizonyították, hogy anomália akkor és csak akkor fordulhat elő, ha $m<M<2m-1$, mivel egész számokról van szó, ebből nagyon könnyen kapjuk: $m>=3$.

Ha m=M, akkor a két memória nem különböző, hanem ugyanazt a memóriát vizsgáljuk kétszer.

Ha N=M, akkor megint nem beszélhetünk anomáliáról, hiszen az $\underline{M}$-ben csak addig van változás, amíg be nem hozzuk az összes lapot, aztán nincs több, és minden új lapot a kicsibe is be kell hozni, így anomália nem állhat fenn.

Marad: ***N>M>m>=3*** Mostantól kezdve ilyen *N,M,m* értékekre fogok hivatkozni, hacsak ezt másképp nem jelzem.

Példa

Ebben a fejezetben példákat fogok mutatni, hogy 2-nél nagyobb anomália elérhető – szemben a [2] állításával, valamint az itt szereplő példákon bemutatom a dolgozat anyagát szemléletesen.

Jelölések

A programot egy laphivatkozási sorozattal fogom jellemezni, amelyben a lapokat számokkal jelölöm meg. A memória aktuális állapotát a következő módon lehet kényelmesen megjeleníteni:

Memória méret: 3.
Hivatkozási sorozat: 1,2,3,4,5.

Memória állapot
1 2 3
4 5

A jelölés értelmezése: Most a memóriában a 3,4,5 lapok szerepelnek, mielőtt az 5-öt behoztuk volna a 2,3,4 volt a memóriában. Az 5 behozásával a 2 kikerült. A lapokat sorfolytonosan írhatjuk le, és mindig azok vannak aktuálisan bent, amelyek alatt nem szerepel semmi, és éppen azokat a lapokat vitték ki, amelyik alá leírjuk őket. Ez a jelölés a FIFO tulajdonságot használja ki.

Az anomáliát úgy vizsgáljuk, hogy veszek egy hivatkozási sorozatot, és megszámoljuk, hogy hány lapcserére van végrehajtásához szükség egy adott memóriaméret mellett, illetve eggyel kisebb memória méret esetén. Ha a kisebbik memóriánál kevesebb lapcsere történik, akkor az anomália fennáll.

1. példa

Az első példa egyben az adott paraméterek mellett egy maximális anomália is lesz, így egy jó lehetőség arra, hogy bemutassam, hogy miről fognak a következő fejezetek szólni.

Legyen a példánkban:

Rendelkezésre álló lapok száma: 7
A nagyobb tár mérete: 6
A kisebbik tár mérete: 5

A hivatkozási sorozat: 1,2,3,4,5,6,7,1,2,4,5,6,7,3,2,4,5,1,7,3,6,2,5,7!,
1,2,3,4,5,6,7,1,2,3,4,5,6,7,1,2,3,4,5,6,7!,
1,2,3,4,5,6,7,1,2,3,4,5,6,7,1,2,3,4,5,6,7,
1,2,3,4,5,6,7,1,2,3,4,5,6,7,1,2,3,4,5,6,7,
1,2,3,4,5,6,7,1,2,3,4,5,6,7,1,2,3,4,5,6,7,
1,2,3,4,5,6,7,1,2,3,4,5,6,7,1,2,3,4,5,6,7.

Erre a sorozatra:

A lapcserék száma a nagyban: 119
A lapcserék száma a kicsiben: 58
Anomália mérték eddig: 119/58 > 2, azaz sikerült 2 fölé vinni az anomáliát, ezzel a [2] sejtése megdölt.

A memóriák állapotának leírása (minden leírt számjegy a jelölésmód miatt egy cserét jelent):

Nagy	*Kicsi*
1 2 3 4 5 6	1 2 3 4 5
7 1 2 3 4 5	6 7 1 2 4
6 7!1 2 3 4	5 6 7 3 2
5 6 7 1 2 3	4 5 1 7 3
4 5 6 7 1 2	6 2 5!1 4
3 4 5 6 7!1	7 3 6 2 5!
2 3 4 5 6 7	1 4 7 3 6
1 2 3 4 5 6	2 5 1 4 7
7 1 2 3 4 5	3 6 2 5 1
6 7 1 2 3 4	4 7 3 6 2
5 6 7 1 2 3	5 1 4 7 3
4 5 6 7 1 2	6 2 5
3 4 5 6 7 1	
2 3 4 5 6 7	
1 2 3 4 5 6	
7 1 2 3 4 5	
6 7 1 2 3 4	
5 6 7 1 2 3	
4 5 6 7 1 2	
3 4 5 6 7	

A példa magyarázata:

Érdemes észrevenni, hogy a felkiáltójellel (!) körülvett rész már folyamatosan ismétlődik: mindkét memóriában ugyanazt a változást idézi elő a hivatkozási sorozat felkiáltójelek közé eső része. Azaz innentől kezdve ugyanez fog történni, azaz 7-et hozok a kicsibe, és 21-et a nagyba a hivatkozási sorozat egy 21 elemű részsorozatával.

Érdekesség, hogy a kicsi úgy változik, hogy minden lap alatt a nála eggyel nagyobb szerepel. Látni fogjuk, hogy ez az, ami az anomália maximális értékét biztosítja, és innen származik, hogy mi a 7, 3, 6, 3, 5-öt akartuk kialakítani. Ez a memória állapot, amely az ismétlődés létrejöttéhez szükséges.

Kezdjük ezzel a hivatkozási sorozattal a programot: 1,2,3,4,5,6
A memóriák állapota így változik ezalatt:

1 2 3 4 5 6 1 2 3 4 5
6

Vegyük észre, hogy mostantól kezdve a nagyobb memória ciklikusan fog változni. Egy ilyen 1-essel kezdődő sorozatot fogok **ciklus**nak nevezni.

Megjegyzés: helymegtakarítás végett csak a memóriában lévő lapok leírásával fogom folytatni a leírást a következő sorokban (így persze az oszlopok helye megváltozhat a jelölésben).

A következő állapotot fogom most elérni a kisebb memóriában: 7 3 6 2 5. Ehhez a később bemutatásra kerülő algoritmust használom: megpróbálok mindig minél nagyobb részt kialakítani a kismemóriában, amely megfelel ennek.

A hivatkozási sorozat tehát így folytatódik: 7.

Ezzel el is akadtam, mert a 3 nem következhet. Nem baj menjünk addig, amíg nem lesz a 3 kint, majd "kint tartom", addig, amíg a 7 ki nem ér, majd hozzuk be őket:
Tehát folytassuk: 1,2,4,5,6,7,3
A memória állapot közben így változott az előzőkhöz képest (első sor):

1 2 3 4 5 6	2 3 4 5 6
7 1 2 3	7 1 2 4 5
	6 7 3

Most itt megint elakadtunk, a 6-ot még nem tudjuk behozni. Használjuk az előbbi trükköt és folytassuk: 2,4,5,1,7,3,6,2,5

5 6 7 1 2 3	4 5 6 7 3
4 5 6	2 4 5 1 7
	3 6 2 5

Ezzel el is értük a célunkat. Most szeretném, ha a nagymemória egy ciklusnál kezdődne.
Ehhez behozok: 7

2 3 4 5 6 7	7 3 4 2 5	(azaz az 1 van kint a nagyból, és így a ciklus kezdeténél vagyunk.)

Ez egy nagyon jó állapot, mivel:
folytassuk a következő sorozatta: 1, 2, 3, 4, 5, 6, 1, 2, 3, 4, 5, 6, 1, 2, 3, 4, 5, 6.
Ekkor ez a változás:

2 3 4 5 6 7	7 3 6 2 5
1 2 3 4 5 6	1 4 7 3 6
7 1 2 3 4 5	2 5
6 7 1 2 3 4	
5 6 7	

Látható, hogy visszaérkeztünk a kiinduló állapothoz. Megkaptam a fenti példánál felkiáltójellel jelölt ismétlődést.

Néhány fontos észrevétel:
-Amíg 7-et hozok a nagyobbikba, addig általában a kisebbikbe 7-5=2-t kell hoznom, azaz azokat a lapokat, amik eredetileg nem szerepelnek benne a hétből, be kell hoznom. Ezalól egyedül az a kivétel, amikor az 1-et hozom be, ilyenkor három lapot kell behoznom a kicsibe. A többlet onnan adódik, hogy olyan lapot viszek ki (a 7-est), amit még nem használtam fel ezen *ciklus* alatt.

-A példában eddig 119 csere volt a nagyban, és 58 csere a kicsiben azaz eddig az anomália mértéke 119/58. Mostantól kezdve – a fenti módon folytatva –, tartható a 21/7=3-as arány, azaz az anomália tetszőlegesen meg fogja közelíteni a 3-at elég hosszú hivatkozási sorozattal.

2. példa

Ha nagy anomáliát akarunk, akkor (mint látni fogjuk) célszerű olyan memória méreteket választani, ahol a különbség csak kicsi. Lehetőleg egy, mint a fenti példán. Az előző példában kihasználtam azt, hogy kis memória mérete páratlan. Később a bizonyítás során látni fogjuk, hogy páratlan esetnél egyértelmű az anomália maximuma a különböző hivatkozási sorozatokra, és az az első példában megadotthoz hasonlóan áll elő.

A kis memória méretének páros esetében ezt nem tudjuk kijelenteni – kereső algoritmussal nagyon sok lehetséges megoldást találtam (ld. az ezzel foglalkozó fejezetben), amelyek egyformán nagy anomália értéket adnak. Ezek közül a következő a leglátványosabb, és ez olyan, amely bármely két memória méret esetén kialakítható. Még páratlan esetben is kialakítható, és ez a megoldás nem sokkal marad el az optimálistól.

Most a kismemóriába *csak csökkenő sorrendben hozok be lapokat*. Ehhez persze egy kicsit trükközni is kell. Ezt mutatnám be a példán:

Rendelkezésre álló lapok száma: 8
A nagyobb tár mérete: 7
A kisebbik tár mérete: 6.

Először az előbb bemutatott módon -a következő fejezetben lévő algoritmussal – érjük el a memóriákban a következő állapotot:

Nagy	*Kicsi*
2 3 4 5 6 7 8	8 7 6 5 4 3

Innentől kezdve a hivatkozási sorozat:
2!,1,2,3,4,5,6,8!,7,8,1,2,3,4,6!,5,6,7,8,1,2,4!,3,4,5,6,7,8.

Erre a darabra a lapcserék így alakulnak:

Nagy	*Kicsi*
2 3 4 5 6 7 8	8 7 6 5 4 3
1 2 3 4 5 6 7	2 1 8 7 6 5
8 1 2 3 4 5 6	4 3
7 8 1 2 3 4 5	
6 7 8	

Ezalatt a lapcserék száma a nagyban: 24
Ezalatt a lapcserék száma a kicsiben: 8
Anomália mérték erre a szakaszra: 24/8=3.

Azaz az elején bármely áron is értük el a kezdő állapotot, mostantól kezdve a fenti hivatkozás sorozatot elegendő sokszor megismételve a 3 tetszőlegesen megközelíthető.

A példa magyarázata

Vegyük észre, hogy a felkiáltójellel (!) megjelölt pontokon a hivatkozási sorozatbar felcseréltem a természetes sorrendet. Erre a sorrendcserére azért volt szükség, mert így tudtan biztosítani, hogy lapok sorszámuk szerint csökkenő sorrendben kerüljenek be a kisebb memóriába. Ez az a trükk, amelyhez folyamodni kellett, hogy a sorrend megmaradjon.

Most a paramétereket, a memória méreteket eggyel nagyobbnak választottam, mint az első példában, de az anomália ugyan akkora maradt. Már ez is jelzi, hogy várhatóan ez a fordított sorrendű megoldás elég jó, de nem elég a maximum biztosításához mindig. Bár hasonló paraméter választás mellett (a kis memória mérete páros, egy az eltérés a memória méretek között), nem sikerült ennél jobbat találni, annak ellenére, hogy ehhez hasonló anomália értéket adó megoldás nagyon sok létezik még ilyen kicsi paraméterek esetén is.

Egy lapsorozat előállítása a kisebb memóriában

Tétel. $N=M+1>m$ esetén tetszőleges lapsorozat előállítható az m-ben megfelelő hosszú hivatkozási sorozat segítségével olyan módon, hogy a nagyobbik memóriában csak ilyen sorrendbe hozok be lapokat: 1,2,3,...,N,1,2,3,....,N,1,2,3,....,N,.... és így tovább.

Definíció. A nagy memóriában a lapok fenti váltakozásának egy 1,2,3,....N-ből álló szakaszát **ciklusnak** nevezem. Beszélhetünk például a hivatkozási sorozatnak egy ilyen **ciklusnyi változást** okozó részéről, ekkor a hivatkozási sorozatnak arra a részére gondolunk, amely egy ilyen ciklusnyi változást okoz a nagyobb memóriában.

Mi is a feladat?

Az, hogy a hivatkozási sorozatom elején szerepeljen az 1,2,3,4,....M részsorozat, mivel ez elégséges feltétele annak, hogy a fenti ciklikus váltakozás kialakuljon. (Azért mert mindig csak egy lap lehet kint a nagyobb memóriából, és az a lap mindig a ciklus következő lapját viszi ki, ha behozom.)

Ezzel persze lapokat kellett behoznom a kis memóriába is. Most a feladat az, hogy akár hogy ügyeskedve – a fenti miatt a nagy memóriát már nem ronthatom el – érjük el az általunk kívánt sorrendet. Azaz a feladat az, hogy generáljak egy hivatkozási sorozatot, amelynek a végére a kívánt állapotot érem el.

Konstruktív módon oldom ezt meg, azaz adok egy algoritmust, amely megoldja a problémát. Az áttekinthetőség érdekében rögtön az algoritmust adom meg, a helyesség bizonyításához szükséges feltételekkel, és a megértéséhez szükséges megjegyzésekkel együtt. Az algoritmusban többször szerepel a "behozom" utasítás, amelynek az a jelentése, hogy a hivatkozási sorozat következő elemét ez adja meg, és azzal nem foglalkozom, hogy M-be eközben milyen lapok kerülnek be.

Először leírom **verbálisan**: (ld. még a korábbi példánál a működést.)
Első lépésben tehát a hivatkozási sorozat kezdődjön 1,2,...M sorozattal. Ezzel mindkét memóriába behozom ezeket a lapokat. Mostantól kezdve *brute force* módon dolgozom. Megpróbálom elejétől kezdve behozni a kívánt lapokat sorrendben a kicsibe. Ha sikerül, akkor készen is vagyunk. Előfordulhat, hogy elakadunk – nem tudjuk a sorozat következő elemét behozni –, mert már bent van valahol a kis memóriában az a lap. Ekkor elkezdünk lapokat behozni a kis memóriába, mondjuk mindig a legkisebb kínlévőt, hogy kivigyük azt a bizonyos lapot. Ha kikerült, akkor kint tartjuk. Ezt megtehetjük, mert $N>=m+1$, azaz legalább két lap van kint a kis memóriából. Most tehát kint tartva a sorozatunk következő lapját behozunk lapokat, addig amíg az eddig sikeresen behozott részsorozat eleje ki nem kerül. Ha kikerült, akkor újra behozzuk, ezzel az eddig sikeresen behozott részsorozat következő tagja kikerül. Ciklusban így folytatjuk addig, míg el nem jutunk az egészidáig kint tartott elakadást okozó laphoz. Azaz mostmár behozhatjuk, és folytathatjuk a kis memóriába behozni kívánt sorrendben következő lappal.

Az algoritmus

Definíció. **behozom: kiírom, mint a hivatkozási sorozat következő lapját**, amely a kívánt hatást el fogja érni. A szóhasználat onnan van, hogy így a programba kapcsoszárójelek közé írt elő és utófeltételeknek úgymond szemléletesebben megfelel.

{ Q = N>M>m és a $b_1,...b_m$-et akarom elérni $\underline{m}$-ben }
(1) Behozom az 1,2,...,M lapokat *–ezzel létrejön a ciklus az $\underline{M}$-ben.*
{ Q és $\underline{M}$=(1,2,...M) és $\underline{m}$=(M-m+1,M-m+2,...M) }
(2) i=0
(3) Ciklus, amíg i<m
{Ciklus Invariáns = $\forall u \in [1,i]$: $\underline{m}_{(u+(m-i))}=b_u$}
(3.1) if $b_{i+1} \in \underline{m}$ *– Azaz a következő most nem hozható be.*
–Megj.: Mivel N>=m+2, ezért biztos, hogy legalább 2 lap kint van mindig
–Megj.: Az is biztos volt, hogy a b_{i+1} előbb megy ki, mint a $b_1,...b_i$
–bármelyike. Ezt a Ciklus Invariáns biztosítja, azzal, hogy ez a sorozat
–szogorúan a végén van. Röviden:
{Ciklus Invariáns} és {b(i+1)$\in\underline{m}$} => {$\exists v \in [1,(m-i)]$:b(i+1)=$\underline{m}$(v) }
(3.1.1) Ciklus amíg $b_{i+1} \in \underline{m}$
(3.1.1.1) A legalább két kínnlévő közül a kisebbiket behozzuk
(3.1.1.2) Ciklus vége
{ b(i+1)$\notin\underline{m}$ és a b1,...bi sorozat még összefüggően bent van, ez az előző feltételből levezethető } és b(i+1)-en kívül van még egy, ami kint van.
(3.1.2) Ciklus amíg b1$\in\underline{m}$
(3.1.2.1) A legalább két kínnlévőből a legkisebb, nem b_{i+1}-t behozom.
(3.1.2.2) Ciklus vége
{ $b_{i+1}\notin\underline{m}$ és a $b_1 \notin\underline{m}$ és $b_2,...b_i$ sorozat még összefüggően bent van }
(3.1.3) j=1
–Most egyszerüen mindig kiviszem a $b_{(j+1)}$-t, a b_j behozásával
(3.1.4) Ciklus amíg j<i+1
(3.1.4.1) Behozom b_j-t *–(ez kiviszi a sorozat következőjét, ha még –nem értünk a bi-hez)*
(3.1.4.2) j=j+1
(3.1.4.3) Ciklus vége
{ $b_{(i+1)} \notin\underline{m}$ és Ciklus Invariáns }
(3.1.5) Elágazás vége
{ Ciklus Invariáns és $b_{(i+1)} \notin\underline{m}$ }
(3.2) b_{i+1}-t behozom $\underline{m}$-be
(3.3) i=i+1
{ Ciklus Invariáns }
(3.4) Ciklus vége
{ $\forall u \in [1,m]$: $\underline{m}_u=b_u$ }

A fenti algoritmus tehát egy tetszőleges részsorozatot előállított $\underline{m}$-ben, bár igaz, nem tudjuk, hogy a cikluson belül hol áll a $\underline{M}$ – azaz melyik a kintlévő lap.

Megjegyzés: Nem igaz, hogy elérhető, hogy az $\underline{m}$-ben és $\underline{M}$-ben tetszőleges variáció (tetszőleges sorozatok) keletkezzenek. Ehhez elég ezt látni:

Lemma. Az utoljára behozott lapnak mindkét memóriában szerepelnie kell.

Példa: $\underline{m} = \underline{(2, 3, 5)}$, és $\underline{M} = \underline{(1, 2, 3, 4)}$. Akár az ötöt, akár a négyet hoztuk be utoljára az nem szerepel a másikban. Tehát, ez a helyzet nem alakulhat ki.

Bizonyítás. Tekintsük az S hivatkozási sorozatot. Legyen az "utoljára behozott lap" a k-adik lap a sorozatban. Ekkor a lemma azt mondja ki, hogy S_k eleme $\underline{m_k}$-nak és $\underline{M_k}$-nak is.
$\underline{m_k} := t(f(S_k, m), m)$ és $\underline{M_k} := t(f(S_k, M), M)$. Nézük meg, hogy az S_k behozása elött milyen esetek vannak:

Definíció. A hivatkozás négy esete:

I. $S_k \in t(f(S_{k-1}, m), m)$ és $S_k \in t(f(S_{k-1}, M), M)$
II. $S_k \in t(f(S_{k-1}, m), m)$ és $S_k \notin t(f(S_{k-1}, M), M)$
III. $S_k \notin t(f(S_{k-1}, m), m)$ és $S_k \in t(f(S_{k-1}, M), M)$
IV. $S_k \notin t(f(S_{k-1}, m), m)$ és $S_k \notin t(f(S_{k-1}, M), M)$.

Most az első eset nem áll fenn. A IV. esetben S_k-t be kell hozni mindkét memóriába, azaz itt teljesül a lemma a S_k feldogozása után. A II. esetnél S_k feldolgozása után $\underline{M_k}$-ba épp most kellett behozni, $\underline{m_{k-1}}$ pedig egyenlő $\underline{m_k}$-val, azaz itt is teljesül a Lemma, hasonlóan a III. esetnél is. ■

Megjegyzés: a hivatkozás négy típusa közül az első típusúval nem is érdemes foglalkozni, az az anomália szempontjából semmilyen szerepet nem játszik, akár el is hagyhatjuk.

Felső korlát

Lemma. Ha behozok N különböző lapot az $\underline{M}$-be, akkor ezalatt legalább N-m lapott hozok be az $\underline{m}$-be.

(A jelölés értelmezése a matematikai modellhez: Egy S_k hivatkozási sorozattal eljutottam $\underline{m_k},\underline{M_k}$ állapothoz, innen a hivatkozási sorozatot folytatva egy i hosszú lapsorozattal elérem, hogy $f(S_{(k+i)},M)$ végére N különböző lap kerüljön hozzá $f(S_k,M)$ bővítéseként, ekkor $\#f(S_{(k+i)},m)$ - $\#f(S_k,m) >= N$-m.)

Bizonyítás: A korábbi az első algoritmusnál szereplő lemma alapján, ennek az N különböző lapnak szereplenie kell az $\underline{m}$-ben is, de abban csak m lap volt maximum kezdetben, tehát N-m lapott biztos behoztunk. ■

Eddig az anomália kapcsán teljesen általános esettel foglalkoztunk, most teggyük meg a következő kikötést: $N:=M+1$. Ez a kikötés [2]-ben is szerepel, és az általánosság elvesztése nélkül megtehető. Látni fogjuk, hogy a fenti lemma miatt, ha ezt a kikötést nem tesszük meg, akkor a későbbiekben ebben a cikkben konstruált anomália mértéke mindenképp nagyobb lesz.

Ez azért hasznos kikötés, mivel miután N-1=M lapot behoztunk $\underline{M}$-be mindig egy lehetséges lap nincs benn, és mindig a legelső megy ki (FIFO), azaz innentől kezdve a nagymemória ciklikusan változik csak. Az ilyen N lapból áll „csere" sorozatot neveztem el *ciklusnak* az előző fejezetben. Azért, hogy könnyebb legyen a lapokra hivatkozni vezessünk be egy új számozást (cimkézést a lapokon): miután M lapot behoztunk, akkor ezek sorszáma legyen az $1,2,...M$, így egy **ciklust** kaptunk folytonos számozással – a kintmaradt lap sorszáma az $M+1$.

A **ciklust** más módon is lehetne definiálni úgy, hogy a lapokon lévő kezdeti sorszámozást nem vesszük figyelembe, hanem ehelyett a nagymemóriába bekerült lapokat az első bekerülésük helyével (sorszámával) egyszerüen újraszámozzuk. ($t(f(S,M),M+1)$ sorozatban szerepel mind az $M+1=N$ lap, és az ebben a sorozotban elfoglalt hely lesz a lap sorszáma mostantól kezdve.) Ezzel a fajta sorszámozással ugyanúgy létrejön a „ciklus" fogalma.

Azaz a lapbevitel sorozatot így lehet jellemezni:

A hivatkozási sorozat: $S := 1,2,..,M, r_{(M+1)}, r_{(M+2)}$, ahol az r sorozat teszőleges.

A lapbeviel sorozat: $f(S,M) = 1,2,...M+1,\ 1,2,...M+1,\ 1,2,...M+1, ...$

A fenti ciklus létrejötét egyébként az előző részben ismertetet algoritmus (1) utasítása biztosítja, így a ciklus az algoritmus lefutása után is megmarad – bár nem tudjuk, hogy épp melyik a ciklus kintlévő lapja, –és az algoritmussal egy kívánt optimális helyzet elérhető.

A következőkben egy *ciklusnyi változás* alatt azt fogom érteni, hogy a nagymemóriában egy ilyen $1,2,....M+1$-el bővül a lapbevitel sorozat. Világos, hogy ha meg tudom mondani, hogy egy ilyen „ciklus" alatt mennyi lapott kell behoznom minimálisan $\underline{m}$-be, akkor ezzel becslést is adtam a limam értékére felülről:

limam <= N/(az $\underline{m}$-be behozott lapok minimális száma egy ciklus alatt.)

Megjegyzés: Miközben egy *ciklusnyi változásról* beszélek, akkor nem foglalkozom azzal, hogy eközben ez a program hivatkozási sorozatában mekkora részt jelent. A hivatkozási sorozat szempontjából van egy pont, ahol elkezdődik (kint van az 1-es) és van egy pont ahol befejeződik (épp kiment az 1-es).

Összefoglalva igaz általános esetben a következő:

Lemma. *limam <= N/(N-m).*

Bizonyítás. Lásd. előző lemma és a fent leírtak. ■

Ezt a korlátot sajnos nem lehet teljesen elérni, de ez egy elég jó korlát. A következő fejezetben, amikor a fenti korlát finomításáról beszélek, akkor igazából azon kivételeket próbálom meghatározni, amelyek miatt nem érhő el ez az arány.

A felsőkorlát finomítása

Tétel. *limam <= (N-1)/(N-m), ha N=M+1>m.*

Megjegyzés: Az előző fejezetben ismertetet lapszámozást (ciklust) fogom használni.

Bizonyítás. Ez több lépésben fog történni.

a.) Lemma. *limam = N/(N-m)* nem állhat fenn:

Bizonyítás.

Amikor egy ciklusnyi változás van *M*-ben, akkor az *N-m* db lapot mindenképp be kell hozni *m*-be. Az, hogy csak ennyit kelljen behozni, elengedhetetlen feltétele, hogy azok a lapok, amelyek most ki fognak kerülni előbb kerüljenek felhasználásra, hivatkozásra a ciklus behozása közben, mint azok a lapok, amelyeket most hozok majd be *m*-be. Ennek szükséges feltétele a következő: *Az éppen kivitt lap sorszáma kisebb legyen az éppen behozottnál.* Világos, ha ez nem állna fenn, akkor a kivitt lapot a ciklus befejezéséhez vissza kellene hozni. Azonban az *1*-nél nincs kisebb sorszámú, azaz amit ez kivisz, azt biztos be kell hozni.

Megjegyzés az optimális megoldás kapcsán: Hasonlóan lehetne okoskodni, a *N*-es sorszámú lappal is, hiszen amikor ez kikerül, akkor biztos olyan lap vitte ki, amely kisebb sorszámú, tehát vissza kell hozni. Plusz egy csere, kivéve akkor, hogyha az *1* vitte ki az *N*-es sorszámút, hiszen ekkor ez a két eset nem különbözik, azaz *1* eset. Ezt a gondolatmenetet még tovább lehet vinni, a *2*-es bejön, akkor csak az *1*-est viheti ki, hogy megfeleljen a fentieknek. Az *(N-1)*-est azonban, csak az *N*-es viheti ki. Ahhoz, hogy ezt a többi lapról is ki tudjuk jelenteni, ahhoz be kellene látni, hogy a lapok egyforma gyakorisággal szerepelnek az *f(S,m)*-ben ha az optimális megoldást tételezzük fel. Ezt pedig még nem sikerült bizonyítani, ezért más módszert fogok használni, de látni fogjuk, hogy a megfelelő paraméterválasztás optimális megoldása viseli ezeket a tulajdonságokat, azaz minden lap a nálánál eggyel kisebb sorszámút viszi ki, így csak az 1 behozása okoz egy plusz cserét.

b.) Lemma. a tétel korlátja tetszőlegesen megközelíthető

Legyen *N* páratlan, *M=m+1, N=M+1* és *N>5*. Ekkor vegyük következő sorozatot:

A= 0, (N-1)/2 mod N, 2(N-1)/2 mod N, 3*(N-1)/2 mod N, ..., (N-1)*(N-1)/2 mod N,*

0, (N-1)/2 mod N, 2(N-1)/2 mod N, 3*(N-1)/2 mod N, ...*

Mivel *N* páratlan, így a legnagyobb közös osztóra fennáll:

(N,(N-1)/2)=(N,N-1)=1.

Ami, mint számelméletből ismeretes, annyit jelent, hogy a fenti sorozat első *N* eleme ki fogja adni *N* teljes maradékosztály rendszerét. Az is igaz, hogy tetszőleges *N* egymás utánit veszünk a sorozatból az is kiadja *N* teljes maradékosztály rendszerét, mivel *(0,1,2, ..., N-1)* teljes maradékosztály rendszer.

A sorozatnak van egy további érdekessége, mégpedig az, hogy az egymástól *N-2* távolságra követő elemek a maradékrendszer szerint egymás rákövetkezői.

Bizonyítás: (a sorozat definícióját felhasználva)

$A_{(i+N-2)}=[A_i + (N-2)*(N-1)/2] \bmod N=[A_i + (N*N-3N+2)/2] \bmod N.$

Használjuk ki, hogy *(N*N-3N)/2 mod N = 0*, akkor kapjuk:

$A_{(i+N-2)}=[A_i + 1] \bmod N.$

Ami éppen a keresett állítás.

Ez egy nagyon jó tulajdonság, hiszen $N-2=m$, ez pedig azt jelent, hogyha ez lenne a lapbevitel sorozat, akkor az éppen behozott lap mindig az $N-2$-vel korábbit vinné ki. Méghozzá úgy, hogy annál eggyel nagyobb lenne ($mod\ N$). Azaz pont az a.) részben leírt tulajdonságot kaptuk, a 0 kivételével minden lap nálánál kisebb sorszámút visz ki. Azt a szépséghibát, hogy ne 0-val kezdődjön a sorozat $N-1$-ig egyszerűen kiküszöbölhető azzal, ha a sorozatot úgy használom fel, hogy minden tagjához hozzáadok egyet.

A fenti optimális tulajdonságot kiadja a következő:

S= (a kezdő helyzet kialakítása a korábbi algoritmussal), $1,2,3,...,N,1,2,3,...,N,$ ...

Ekkor a lapbevitel sorozat így alakul: $f(S,m)_{(Km+i)} = A_i$.

A kezdő helyzet: $\underline{m}= A_3+1,\ A_4+1,\ ...\ A_N+1,$ és Km azon lapcserék számát jelenti, amelyek a kezdő helyzet kialakításához vezettek. Ezalatt KM-t cseréltünk a másik memóriában.

A korábbi algoritmus kapcsán megjegyeztük, hogy nem lehet tudni, hogy milyen fázisban hagyja az $\underline{M}$-t. A paraméter választás miatt az $\underline{M}$ ciklikusan fog változni, tehát igazából csak azt kell elérnem, hogy a ciklus elejére kerüljön, azaz az 1 legyen kint a memóriából. Ez azonban nem jelent gondot, mivel egyszerűen így folytatom a hivatkozási sorozatot az algoritmus lefutása után: $1,2,....N$. Ekkor $\underline{M}$-ből kint lesz az 1 (az N viszi ki), az $\underline{m}$ pedig úgy viselkedik, mintha tényleg behoztunk volna lapokat az $\underline{M}$-be. Azaz innentől kezdve mindig N-t cserélünk $\underline{M}$-ben, amíg csak 2 az $\underline{m}$-ben, vagy 3-t amikor az 1 jön be. Azaz fennáll a következő:

$(KM+N+n*X)/(Km+3+n*Y) = am.$

Ha n tart a végtelenbe: $X/Y = limam.$

Tekintsünk egy akkor részt, amikor N lapot hozunk be $\underline{m}$-be. Kérdés, eközben hány ciklust tud megtenni a nagymemória: $(N-1)/2$ darabot. Indoklás: Az 1 behozása $+1$ cserét okoz. A többi esetben marad az optimális $N-m=2$ behozás, azaz:

$X/Y = \{(N-1)/2*N\} / N = (N-1)/2 = anomália.$

A pontos értékét, $(N-1)/2$, a d.) pontban fogom kiszámítani. Előbb azonban térjünk ki a paraméterek megválasztására.

c.) Tehát láttuk, hogy a fenti elérhető N páratlan, $M=m+1,\ N=M+1$ esetben. Most lássuk be:

Lemma. Ha m kisebb, akkor csak rosszabb megoldáshoz juthatunk.

Bizonyítás. Triviális az egy ciklus alatt behozott lapok száma alapján. Az előbb biztosítottuk, hogy:

$limam1=(N-1)/2.$

Az egy ciklus alatt behozott minimális lapszámra a „Felső korlát"-ot alkalmazva:

$limam2 <= N/(N-m).$

Azaz ha most kisebb m-et használnánk, akkor:

$N/(N-m) <= N/3$, mivel tudjuk, hogy $N>3$, ezért fennáll, hogy:

$N/3 < (N-1)/2 = limam1$, azaz

$limam1 > limam2.$

Itt a c.) pontban annyit láttunk be, hogy ha $N=M+1$ teljesül és ez páratlan, akkor nem érdemes kisebb m-mel kísérletezni, mert akkor nem fogunk jobb anomáliát kapni.

d.) Határozzuk meg pontosan az anomáliát:

Vágjuk át a hivatkozási sorozatot azon a ciklus kezdeténél, ahol az *1* kint van az $\underline{m}$ memóriából. (Ez biztos fenn fog állni, hiszen egyrészt az optimális csereszám eléréséhez csak azokat tudom behozni, amelyek az előző ciklusban kimentek. Másrészt az az eset, amikor az előző ciklushoz hozzácsapjuk az *1* behozását, az lényegében ugyan ez a megoldás, hiszen az *1* így is plusz cserét okozott, és ebben a ciklusban azok vannak kint, amik így is-úgy is kint vannak.)

A következő ciklusban a legelső behozott lap az *1* lesz.

Biz: Ha nem az lenne, akkor is mindenképp a ciklus elindulásakor be kell hozni, hiszen ez a ciklus induló eleme, azaz, miután behoztuk, még mindig be kell hozni az összes ekkor kintlévő lapot, mivel azokat még nem használtuk fel a ciklus befejezéséhez, tehát minden lap, amit az *1* előtt hozok be, plusz egy cserét okozna.

I. eset: m=*páratlan*, $M=m+1$, $N=M+1$.

Vágjuk át a hivatkozási sorozatot azon a ciklus kezdeténél, ahol az *1* kint van a kicsi memóriából. Tekintsünk egy akkora részt, ami alatt N lapot hozunk be $\underline{m}$-be. Kérdés, eközben hány ciklust tud megtenni a nagymemória:

$(N-1)/2$ darabot. Indoklás: Az *1* behozása $+1$ cserét okoz. A többi esetben marad az optimális $N-m=2$ behozás, ha a korábban felírt korlátot alkalmazzuk. Tehát legjobb esetben:

$(X/Y) = \{(N-1)/2*N\} / N = (N-1)/2 =$ *anomália.*

Azaz, *anomália* $= (N-1)/2$ ezen a szakaszon (itt nem határérték). Fent konkrétan megmutattuk, hogy ezt hogyan lehet elérni, és látszott, hogy ezt akárhányszor lehet ismételni. Azaz: *limam* $= (N-1)/2$.

További észrevétel: az *1* nem lehet benne a kicsi memóriában. Indoklás:

Ha nincs kint, az azt jelenti, hogy az előbb a ciklus alatt harmadiknak hoztuk be $\underline{m}$-be, ez azonban hasonló probléma az a.) pontban leírthoz, hiszen az *1*-et amikor be kell hoznunk, akkor előtte még *2* másikat is be kellene hoznunk, de amit ezek pluszban kivisznek, azokat még nem használtuk fel, azaz ebben a ciklusban $+2$ lapcserénk lesz. Azaz egy teljes ciklust elvesztünk N lap behozása alatt, és az $N+2$ lap behozásával így is kivisszük az *1*-est.

II. eset: m=*páros*. Itt az okozza a problémát, hogy az $N-1$ páratlan, nem osztható *2*-vel (*0.5* lapcserét nem lehet értelmezni a fenti képletekben.) Igazából most azt kell megmutatni, hogy a fent elért hányadost itt nem lehet javítani.

Bizonyítás: a fenti lemma biztosítja, hogy $N-1$ lap behozásával kivittük az *1*-et. Megint ugyanazok a feltételek, mint a b.) pontban. De most $N-1$ cserét tekintünk a kicsi memóriában, és ezalatt a nagyban: $(N-2)/2$ ciklus lehet. Azaz:

$Anomália = (X/Y) = (N*(N-2)/2) / (N-1).$

Ismert a következő a számtani, mértani közepekre: $k>4$ *és* $k-2 \neq k$ miatt

$(k+k-2)/2 > \sqrt{(k*(k-2))}$

$(k-1)*(k-1) > k * (k-2)$

$(k-1) > k * (k-2) / (k-1).$

Ezt felhasználva: *Anomália < (N-1)/2.*

Észrevétel: Bár már fent bizonyítottam, azért érdemes most itt is megnézni, hogy az az esetet, hogy az *1*-et másodikként hoztam be (most már harmadikként kellene) nem jelent semmilyen szempontból sem jobb megoldást. Ekkor amíg az *1* kikerül *N* cserénk lesz a kicsiben. A nagyban maximum *(N-4)/2 +2* ciklus. Azaz *anomália= (N-2)/2*. A sejtés az, hogy ennél jobb arányt nem is lehet elérni, annyi idő alatt, amíg behozok *N* lapot a kicsibe.

Néhány szó az m páros esetről

Az előbb láttuk, hogy ebben az esetben nem lehet akkora mértékű anomáliát elérni, mint *m* páratlan esetén. Ennek ellenére érdemes egy kicsit elidőzni ennél. Az *(N-2)/2*-es arányt a páratlan esetnél – mint maximumot – nem sikerült bizonyítani, de a számítógépen futtatott maximum kereső programmal nagyon sok lehetséges hivatkozási sorozatot találtunk, amelyekkel ezt az értéket el lehetett érni. A leglátványosabb ezek közül, ha a kismemóriába csak csökkenő sorrendbe hozok be lapokat. Ehhez persze egy kicsit trükközni is kell, ezt most be is mutatnám:

Azt tudjuk, hogy csak *N>=5* esetén létezik az anomália, páros esetben *N>=6*.

Legyen a kiinduló állapot a nagymemóriában *(M)*: *2 3 .... N.*

A kicsiben *(m)*: *N N-1 .... 3.*

Ez elérhető: a kicsiben kialakítható az állapot a korábban megadott algoritmussal, a nagyban pedig a ciklus megfelelő fázisára lehet állni hasonló módon, mint ahogy azt a páros esetnél tárgyaltuk.

Azt akarjuk, hogy a memóriák állapota innentől kezdve "szép" legyen azaz:

f(S,M) = ... 1,2,...,M+1, 1,2,...,M+1, 1,2,...M+1, ...

f(S,m)= .... M+1,....,2,1, M+1,...2,1,....

Innen *2*-vel folytatódik a hivatkozási sorozat. Abban áll a trükk, hogy közben semmit sem hozok be a nagyba. Innen így folyatódik a hivatkozási sorozat: *1,2, ... N-2*. Ekkor az *N* fog következni! (A kicsiből csak most kerül ki az *N-2* -es lap.) Majd befejezzük a sorozatot: *7,8*.

Ekkor a következő állapotban vagyunk:

A nagy: *M=2 3 .... N.*

A kicsi: *m=N-4,... 2,1,N,N-1.*

Innen hasonlóan lehet folytatni: *1,2,...N-4, N-2 (!), N-3, N-2, N-1, N*. A trükközés arra irányul csak, hogy amikor be kell hozni új lapot a kicsibe, akkor előbb kell behozni a nagyobbat – viszont ez csak akkor tehető meg plusz lap behozása nélkül, ha a kiviendőket már felhasználtuk. Amikor az *1*-est kellett behozni, akkor eggyel több lapot hozunk be, mint a páros esetben.

Ezzel a módszerrel az anomália mértéke – mint fent már leírtuk: *(N-2)/2*.

Ez bár nem olyan egyszerűen leírható, mint a páros esetnél leírt hivatkozási sorozat. Megvan az az előnye, hogy ezt egymáshoz képest szabadon választott M,m értékeknél használni lehet, és elég jól közelíti a mindenkor fennálló optimális esetet. Például *N=M+1=m+2* páratlan esetnél az optimum *(N-1)/2* volt. A fenti módszer csak *1/2*-el tér el tőle.

Számítógépes szimuláció eredményei

Molnár Mihály algoritmusa

Mint az a szakdolgozat bevezetőjében szerepel, Molnár Mihály készített egy egyszerű programot arra, hogy vizsgáljuk az anomália mértékét véletlen körülmények között. Igazából ez volt az a program, amely a szakdolgozat megírásához vezetett, mivel egészen lefuttatásáig nem is volt sejthető, hogy az anomália mértéke meghaladhatja a 2-t. A meglepő az volt, hogy egy véletlenszerű algoritmus is milyen nagy anomália értékeket generált.

A program eredeti ötletét a genetikus algoritmusok futási eredményének vizsgálata adta a szerzőnek (genetikus kiserleteiből arra is következtetett, hogy a memória méreteket nagynak és egymáshoz közelinek kell venni). Kisérleti célokra meg is írta, majd megdöbbenve tapasztalta, hogy a programot elindítva néhány perc alatt már meg is döntötte az anomáliára vonatkozó sejtést. Itt ebben a dolgozatban már az algoritmus végső, letisztult változatát adom meg, amellyel a lenti táblázat készült.

Az algoritmus: (A hivatkozás I. típusával itt sem foglalkozunk)

(1) **k=0**
(2) **Ciklus amíg #mk<=100000**
(2.1) **Ha van olyan lap, ami csak a kicsiben van benn**
(2.1.1) **i=az ilyen lapok közül az egyik.**
(2.1.2) **Különben: i=véletlenszerűen egy lap, ami nincs bent a kicsiben. (#mk+1)**
(2.2) **A k+1. lap a hivatkozási sorban az i**
(2,3) **k=k+1.**
(2.3) **Ciklus vége**

Kis magyarázat: (2.1)-es sor egyszerűen annak kihasználása, hogy van lehetőség arra, hogy a nagyba hozzunk be lapot, de a kicsibe ne.

(2.1.2)-es sorban a feltétel arra szolgál, hogy minél előbb közelebb kerüljünk ahhoz a helyzetet, amikor a kicsiben olyan lap van, ami a nagyban nincs. Nem szabadna azonban azt írnunk, hogy "olyan lapot, ami nincs bent a nagyban", mivel ezzel az anomália kialakulásának még a lehetőségét is kizárnánk, mivel a nagymemóriában ugyan azok a lapok cserélődnének ciklikusan.

(2.2)-es sorban a hivatkozási sorozat hatására mindenképp kell, hogy lapcsere történjék.

A program teljes szövege megtalálható a dolgozat végén.

Néhány futási eredmény: A táblázatban az utolsó 1000 lapbevitelnél talált anomália értékek átlaga szerepel, minden esetben addig futattuk az algoritmust, amíg $\#mk = 100.000$ nem volt.

N	M	m	Az utolsó 1000 am átlaga	Megjegyzés
5	4	3	0.740338	
6	5	4	0.870723	
7	6	5	1.016960	
8	7	6	1.164179	
9	8	7	1.317468	
10	9	8	1.446885	
20	19	18	2.916815	Ekkor értékeknél már a véletlen algoritmusnál is
20	19	17	1.424570	előáll az anomália, még akkor is, ha $M-m>1$.
20	19	16	0.861790	
100	99	98	14.276974	
100	10	9	1.012291	Itt érdekesség, hogy viszonylag nagy lapszámnál
100	20	19	1.169923	is előáll az anomália.
200	199	198	26.820116	Látható, hogy az optimálist viszont nem közelíti jól az algoritmus.

Kereső algoritmus: maximális anomália

a.) Anomáliát okozó ismétlődés keresése

Ezt a programot több változatban is megírtam. Az a.) változat listája megtalálható a dolgozat végén.

A program célja az volt, hogy tetszőleges állapotból elindulva minél nagyobb anomália értékeket érjek el. A programot azért hoztam létre, hogy bármely paraméter választás mellett tudjam vizsgálni az anomáliát. Igazából fő célja, hogy azt az esetet tüzetesebben megnézzük, amikor $N=M+1=m+2$, és m páros (ld. "Néhány szó az *m* páros esetről" című fejezetben.)

Az algoritmus lényege, hogy az összes lehetséges kezdő állapothoz generáltam az összes hivatkozási sorozatot, amely egy adott korlátnál rövidebb, és maximum keresést hajtottam rajta végre a maximális anomália (itt minimális lapcserék száma) megtalálásához.

Természetesen így egy kicsit nagy lett volna a futási idő, így a hivatkozási sorozatok közül kizártam azokat, amelyek nem okoznak változást egyik memóriában sem – ez, mint láttuk a *hivatkozás négy esete* közül az I., amely nem játszik az anomália szempontjából semmilyen szerepet sem.

Az algoritmus paraméterei:

n –a lapok száma, a nagymemória méretet ennél eggyel kisebbnek vesszük
m –a kismemória mérete
mélység – Az a mélység, ameddig keresni akarunk nagy anomália értékeket
korlát –A kiírás csak akkor lesz, ha már elérte az anomália ezt az értéket. A kezdeti fölösleges kiírások elkerülése a szerepe.

Az algoritmus: (test3.c)

```
korábbi:=korlát
Kezdetben nincs lap a memóriában
level[]:=0        (a számlálólánc, hogy mi az akt hivatkozási sorozat)
l:=1
Állapot elmentése
Ciklus amíg l>=1        (amíg még lehet léptetni)
        Ismételd:
```

```
            kovm := ++level[l]
        Addig amíg kovm-et nem tartalmazza a memória
        Ha kovm > n akkor:
            level[l]:=0;        (mivel túlfutottunk)
            Állapot Visszatöltése
            --l;
        Különben
            Állapot elmentése
            A lapcsere végrehajtása, anomália (lapcserék) számítása
            Ha az anomália>korábbi akkor
                korábbi:=anomalia
                kiírás hogyan értük el
            l++;
            Ha l>=mélységi korlát akkor
                --l;
                Visszalépés;
    Ciklus vége
```

Ezúttal eltekintek az elő- és utófeltételek megadásától.

Bár *(N-2)/2*-es arányt a páratlan esetnél – mint maximumot – nem sikerült bizonyítani, de ezen algoritmusokkal végzet kísérletezgetés arra enged következtetni, hogy valóban ez a korlát.

A bizonyítás nehézségét az okozza, hogy nagyon-nagyon sokféleképpen el lehet érni ezt a korlátot, és előfordulhat, hogy a hivatkozási sorozat egy-egy szakaszát ha nézzük, akkor valóban megtörténhet, hogy jobb eset is lehetne, ha sokáig tudnánk tartani azt a lapcsere arányt. Mint a korábban a felső korlát bizonyításánál a **d.)** pont **II.** részében megmutattam, hogy a páros eset csak rosszabb lehet, mint a páratlan, de ez nem elég annak bizonyításához, hogy a fenti értéket nem lehet-e meghaladni.

b.) Kezdőhelyzet kialakítása

Az a.) részben ismertetett algoritmussal egy optimális helyzetből való indulást vizsgáltam. A dolgozat első algoritmusánál láttuk, hogy bizonyos esetek kivételével ez a kezdő helyzet el is érhető. A dolgozatban szereplő algoritmus azonban nem a legrövidebb hivatkozási sorozatot adja meg, amellyel ez a helyzet elérhető. Gyakran lényegesen rövidebb hivatkozási sorozat létezik, mint amit a korábban adott algoritmus előállít. Megjegyzés: Ettől még szükségünk volt arra az algoritmusra, mivel az konstruktív bizonyítása volt annak, hogy az általunk kívánt helyzet elérhető.

Ha azt akarjuk vizsgálni, hogy hogyan lehet elérni a kezdőhelyzetet, akkor az a.) pont algoritmusa kis módosítással erre is alkalmas. Mindössze:

1. Az elején nem üres memóriával kell kezdeni, hanem abból az állapotból, ahová azzal jutottunk, hogy feltöltöttük a nagyobb memóriát.
2. Ott ahol az anomáliát vizsgáljuk, ott azt kell megnézni, hogy elértük-e már a kívánt helyzet.

Még egy kicsit módosítva a legrövidebb hivatkozássorozat megtalálására is lehet használni. De ez az algoritmus betöltötte teljesen a funkcióját azzal, hogy mutatott ellenpéldát arra, hogy a dolgozat elején adott algoritmus nem a legrövidebb sorozatot adja.

Az eredmények összefoglalása

Eredmények:

A dolgozat a FIFO anomáliáját vizsgálta részletesen, de a maximum elérhetőségének módjai más lapcseréléses algoritmusra is érvényes. A FIFO esetén beláttuk, hogy csak egy tetszőlegesen megközelíthető értékről beszélhetünk az anomália szélsőértéke kapcsán.

A két paraméter *(m,M)* tetszőleges megválasztása esetén az anomáliára fennáll az *N/(N-m)* felső korlát, amelyet soha nem lehet elérni. Nyilvánvalóvá vált, hogy az anomália szempontjából csak olyan paraméterekkel érdemes foglalkozni, amikor $N>M>m>=3$. Továbbá [2]-ből ismert a következő, az anomália fennállásához szükséges feltétel: $m<M<2m-1$.

Mi vizsgálódásainkban az $N=M+1$ kikötést tettük. Ekkor egy rögzített M érték esetén akkor legnagyobb az anomália, ha a kis memória méretét a lehető legközelebbinek vesszük a nagy memória értékéhez. Azaz: $M=m+1$.

Ekkor két esetet különböztethetünk meg:

1. m páratlan esetén fennáll az *(N-1)/(N-m)* korlát az anomáliára, és ez a korlát tetszőlegesen megközelíthető.
2. m páros esetén is fennáll a fenti korlát, de nem érhető el. A sejtés az, hogy ekkor a pontos érték: *(N-2)/2*. Ennek elérésének módját megadtam.

A páros esetnél megadott módszer tetszőleges paraméterválasztása esetén nagy, a maximumhoz közeli, anomália értékek előállítására alkalmas. Természetesen a páratlan esetnél alkalmazott módszer – amikor használható – mindig nagyobb anomáliát okoz.

Az algoritmusokról szóló résznél Molnár Mihály algoritmusa, és futási eredményei jól mutatják, hogy a hivatkozási sorozatra adott egyszerű kikötéssel olyan hivatkozási sorozatokat kapunk, amelyek egy nagy anomália értékhez konvergálnak.

Az *N/(N-m)* korlátot általánosan használhatjuk az anomália felső korlátjának becslésére olyan helyeken, ahol ez szükséges. Az azonban, hogy mit érhetünk el tetszőleges paraméter értékeknél – még nyitott kérdés. A dolgozat csak a lehetséges legmagasabb érték meghatározását tűzte ki célul.

Programok

Kereső program: maximális anomália

```
/* test.c */
#include <stdio.h>

#define n 6
#define m 4
#define melyseg (20)
#define korlat 2
/* n - lapok szama,
   m - kismemoria merete
   melyseg - Az a korlat, ameddig el megyunk egy kiindulo helyzetbol
   korlat - A korlat alatti anomalia ertekeket nem irjuk ki
*/

int mem[m];       /* Az eppen vizsgalt memoria allapot */
int aktm=0;       /* A behozando lap helye */
int kovm=0;     /* Az a lap, amit be fogunk hozni */
int level[melyseg];  /* A backtrack algoritmus pointerei */
int l;            /* a szint szam */
int kor;  /* a tenyleges korlat erteke a fenti konstans alapjan */
int szamlalo;   /* Cserek szama */
int segedsz;    /* behozas helye */
struct szintek {
        int mem[m];       /* a memoria allapot */
        int aktm;         /* Az elozo behozas helye */
        int szamlalo;   /* Az eddigi lapcserek szamlaloja */
        int segedsz;      /* A kovetkezo behozas helye */
        };
struct szintek sz[melyseg+1];       /* Itt tartom nyilvan, hogy milyen allapotba
                                       kell visszalepni */

int van;  /* Mar szerepel a memoriaban */
int ok;

/* A lap mar szerepel-e a memoriaban (kovm) */
int vanmar()
{
        int i;
        van = 1;
        for( i=0; i<m; i++ )
                van = van && mem[i]!=kovm;
        return (!van);
}

/* Az aktm-et behozzuk a memoriaba, ha meg nem volt bent */
void cserem()
{
        int i;
        van = 1;
        for( i=0; i<m; i++ )
                van = van && mem[i]!=kovm;
        if(van)
        {       mem[aktm] = kovm;
                aktm = (aktm+1)%m;
        }
}
```

```
/* Megvizsgalja, hogy van-e anomalia, a megadott korlatnal nagyobb,
   es ha van, akkor kiirja, hogy hany csere tortent a memoriakban,
   a memoria allapotat, es hogy hogyan ertuk el */
int vege()
{
        int i;
        ok=1;

        if (kor<=szamlalo*(n+1)+segedsz)
        {
                kor=szamlalo*(n+1)+segedsz;
                printf("Vege:%d:%d\n",szamlalo,segedsz);
                for(i=0;i<m;i++)
                {
                        printf("%2d",mem[(aktm+i)%m]);
                }
                printf("   ");
                 for(i=1;i<=l;i++)
                        printf("%2d",level[i]);
                 printf("\n");
        }
}

/* A backtrack celjabol az allapotot elmenti */
void Elment( int l )
{
        int i;
        for(i=0;i<m;i++)
        {
                sz[l].mem[i]=mem[i];
        }
        sz[l].aktm = aktm;
        sz[l].szamlalo=szamlalo;
        sz[l].segedsz=segedsz;
}

/* Az allapot helyreallitja visszalepeskor */
void Vissza( int l )
{       int i;
        for(i=0;i<m;i++)
        {
                mem[i]=sz[l].mem[i];
        }
        aktm=sz[l].aktm;
        szamlalo=sz[l].szamlalo;
        segedsz=sz[l].segedsz;
}

/* Csere volt, a pointereket leptetni kell */
leptet()
{
        int j;
        j = kovm;
        kovm = segedsz;
        while( vanmar() )
        {
                segedsz++;
                if (segedsz>n)
                {
```

```
                        segedsz = 1;
                        szamlalo++;
                }
                kovm = segedsz;
        }
        kovm=j;
}

main()
{
        int i;
        l=1;
        kor=korlat*(n+1);
        szamlalo=0;
        segedsz=1;
        for(i=0;i<m;i++)            /* mem nullazasa */
                mem[i]=0;
        for(i=0;i<melyseg;i++)              /* a szamlalo lanc 0 */
                level[i]=0;
        printf("Program started!\n");

        /* ------------ Fo program ---------------- */
        Elment(l);
        while(l>=1)
        {
                if(l<=1) /* Sima kiiras unalom ellen */
                {
                 printf("(%d,%d): ",l,level[l]);
                 for(i=0;i<m;i++)
                 {
                        printf("%2d",mem[(aktm+i)%m]);
                 }
                 printf("    ");
                 for(i=1;i<l;i++)
                        printf("%2d",level[i]);
                 printf("\n");
                }

                kovm=++level[l];        /* a szamlalolanc leptetese */
                while(vanmar()) kovm=++level[l];
                if (kovm>n)
                {                               /* egy szinttel vissza kell lepni */
                        level[l]=0;
                        l--;
                        Vissza(l);
                } else
                 {
                        Elment(l);      /* nem kell visszalepni */
                        cserem();
                        leptet();       /* a segedsz es szamlalo leptetese */
                        vege();         /* <---------- */
                        l++;                    /* egy szintet elore megyunk */
                        if(l>=melyseg)
                        {       l--;
                                Vissza(l); /* Ha vege, vissza lepes */
                        }
                 }
        }

        printf("Vege.\n");
```

```
}
```

Molnár Mihály programja

```
/* or.c */
#include "or.h"

int S=20;
int MEM1=18;
int MEM2=19;
unsigned char *set1;
unsigned char *set2;
void main()
{
 register unsigned char x,t,h1,h2;
 register unsigned char l1,l2,k1,k2;
 long int in1,in2;
 long int cou;
 float f;
 unsigned int dd;
 FILE *p,*q;

 if((q=fopen("tabla.txt","w"))==NULL) abort();

 if((p=fopen("or.par","r"))!=NULL)
 {
        x=fscanf(p,"%d,%d,%d",&S,&MEM1,&MEM2);
        if(feof(p)) {fclose(p); printf("End\n"); exit(0);}
        if(x!=3) {fclose(p); printf("ParFileErr!\n"); abort();}
 }

while(1)
{

 cou=1000001;
 f=0; dd=0;
 CLEARUP;
 INIT;
 while(cou)
 {
        /*vegigmegyunk set1 lancolasan*/
        x=set1[0];
        while(x!=255)  /*amig nem a vegelem*/
        {
         if(NOT_IN(2)) /*nincs benne set2-ben?*/
         {
                /*akkor behozzuk*/
                INSERT_FIRST(2);
                /*ha megtelt set2, akkor az utolso elemet kidobjuk*/
                DROP_LAST(2);
                break;
         }
         x=set1[x]; /*nezzuk a kovetkezot*/
        }

        if(x==255) /*set1 minden eleme set2-nek is*/
        {
         /*veletlenszeruen behozunk egyet*/
         /*ami valamelyikben nincs benne*/
```

```
        t=RND(S)+1;
        while(set1[t]) t=RND(S)+1;
        x=t;

        if(NOT_IN(2)) /*nincs benne a set2-ben?*/
        {
                /*akkor behozzuk*/
                INSERT_FIRST(2);
                /*ha megtelt set2, akkor az utolso elemet kidobjuk*/
                DROP_LAST(2);
        }

        if(NOT_IN(1)) /*nincs benne a set1-ben?*/
        {
                cou--;
                /*akkor behozzuk*/
                INSERT_FIRST(1);
                /*ha megtelt set1, akkor az utolso elemet kidobjuk*/
                DROP_LAST(1);
        }

       }
      if(cou<1000)
      {
       f+=(float)in2/(float)in1;
/*    printf("\t%lf ",f);
      printf("\t %ld %ld",in1,in2);
      printf("\n");
*/    dd++;
      }
 }

 f/=(float)dd;
 printf("S: %d M: %d m: %d",S,MEM2,MEM1);
 printf("\t%f ",f);
 printf("\t%ld %ld",in1,in2);
 printf("\n");

 fprintf(q,"%d\t%d\t%d",S,MEM2,MEM1);
 fprintf(q,"\t%f ",f);
 fprintf(q,"\t%ld %ld",in1,in2);
 fprintf(q,"\n");
 fflush(q);

 x=fscanf(p,"%d,%d,%d",&S,&MEM1,&MEM2);
 if(feof(p)||bioskey(1)) {fclose(p);fclose(q);printf("End\n"); exit(0);}
 if(x!=3) {fclose(p);fclose(q); printf("ParFileErr!\n"); abort();}

}

}
```

/* or.h */

```
#include <stdio.h>
#include <mem.h>
#include <bios.h>

/* RANDOM FUGGVENY
```

```
#include <stdlib.h>
#define RND(X) rand()%(X)
*/

/* RANDOM FUGGVENY - GYORSABB */
#include "r250.h"
#define RND(X) r250n(X)

#define CLEARUP if(set1!=NULL) free(set1); if(set2!=NULL) free(set2);

#define INIT \
 l1=MEM1+1; l2=MEM2+1;                                          \
 set1=(unsigned char*)malloc(S+1);                  \
 set2=(unsigned char*)malloc(S+1);                  \
 if(set1==NULL||set2==NULL) abort();    \
 setmem(set1, S+1, 0); /*clear set*/    \
 setmem(set2, S+1, 0); /*clear set*/    \
 set1[0]=255; k1=in1=h1=0;                                      \
 set2[0]=255; k2=in2=h2=0;                                      \


/* TOKEN PASTING EXAMPLE
#define VAR(i,j) (i##j)
the call VAR(x,6) would expand to (x6).
*/

#define NOT_IN(NO) !set##NO[x]

#define INSERT_FIRST(NO) \
                set##NO[h##NO]=x;       \
                set##NO[x]=255;         \
                h##NO=x;        \
                k##NO++; in##NO++;      \

#define DROP_LAST(NO) \
                if(k##NO>=l##NO)                                \
                {                                       \
                 t=set##NO[0];                                  \
                 set##NO[0]=set##NO[t];         \
                 set##NO[t]=0;                  \
                 k##NO--;                       \
                }                                       \
```

Irodalomjegyzék

Szorosan kapcsolódó anyagok:

[1] IVÁNYI ANTAL: Operációs Rendszerek elôadás jegyzetei (nem publikált), ELTE TTK, HUNGARY. Info: *EMAIL: TONY@ludens.elte.hu*

[2] BELADY, R.A. NELSON, G.S. SHEDLER. An Anomaly in Space-Time Characteristics of Certain Programs Running in a Pagin Machine. *Comm. of the ACM, 12*, 6, June 1969, 349-353.

[3] RANDELL, B., KUEHNER, C.J. Dynamic storage allocation systems. *Comm. ACM 11*, 5 (May, 1968), 297-305.

[4] BELADY, L.A. A study of replacement algorithms for a virtual storage computer. *IBM Syst. J. 5*, 2 (1966), 78-101.

[5] FINE, G.H., JACKSON, C.W, MCISAAC, P.V. Dynamic program behavior under paging. Proc. 1966 ACM Nat. Conf., Thompson Book Co., Washington, D.C. 223-228.

[6] O'NEIL, R.W. Experience using a time-sharing multiprogramming system with dynamic address relocation hardware. Proc. AFIPS 1967 Spring Joint Comput. Conf. Vol 30, Thompson Book Co., Washington, D.C., pp. 611-621.

[7] BRAWN, B. GUSTAVSON, F. An evaluation of program performance on the M44/44X System, Pt. I. IBM Res. Rep., RC-2083, IBM Corp., 1968. Also as Program behavior in a paging environment, Proc. AFIPS 1968 Fall Joint Comput. Conf. Vol.33,Pt.2, Thompson Book Co., Washington, D.C., 1019-1032.

[8] GALVIN, S.: Operating System Concepts. Addison-Wesley Publication Company, 1994. pp. 312-318.

Egyéb irodalom:

[9] VAN-DEN-BERG, J., GANDOLFI, A.: LRU is better than FIFO under the independent reference model. *Journal of Applied Probability 29* (1992), no. 1, 239-243.

[10]SEIDMAN, T.I.: ``First come, first served'' can be unstable! *IEEE-Trans.-Automat.-Control 39* (1994), no. 10, 2166-2171.

[11]GELENBE, E., GLYNN, P., SIGMAN, K.: Queues with negative arrivals. *J.-Appl.-Probab. 28* (1991), no. 1, 245-250.

[12]SZTRIK, J.; TOMKO, J.: Multiprogramming with inhomogeneous programs. *Alkalmaz.-Mat.-Lapok 8* (1982), no. 3-4, 285-296.

[13]ASZTALOS, D.: Application of finite source queueing models for computer systems. *Alkalmaz.-Mat.-Lapok 5* (1979), no. 1-2, 89-101.

[14]RIGHTER, R., SHANTHIKUMAR, J.G.: Extremal properties of the FIFO discipline in queueing networks. *J.-Appl.-Probab. 29* (1992), no. 4, 967-978.

[15]BRANDWAJN, A., HERNANDEZ, J.A.: A study of a mechanism for controlling multiprogrammed memory in an interactive system. *IEEE-Trans.-Software-Engrg. 7* (1981), no. 3, 321-331.

Tárgymutató

A HIVATKOZÁS NÉGY ESETE, 22
AKTUÁLIS ÁLLAPOT, 12
ALGORITMUS, 21
AM, 12
ANOMÁLIA, 5; 12

BÉLÁDY, 6
BÉLÁDY ANOMÁLIA, 7

CIKLUS, 17; 20; 23
CIKLUSNYI VÁLTOZÁS, 23
CSÖKKENŐ SORRENDBEN, 18

ESETEK, 14

F: LAPBEVITEL SOROZAT, 11
FELSŐ KORLÁT, 23
FIFO, 5

GENETIKUS ALGORITMUS, 8

H: „FEJ" OPERÁTOR, 11
HIVATKOZÁSI SOROZAT, 11

IGÉNY SZERINTI LAPOZÁS, 5

KIS MEMÓRIA, 12

LAPBEVITEL, 11
LAPCSERÉLÉS, 5
LAPCSERÉLÉSES SZÁMÍTÓGÉP, 6
LAPOK, 11
LAPOZÁS, 5
LIMAM, 13
LRU, 6

M44, 6
MEMÓRIA, 5
MEMÓRIÁK ÁLLAPOTA, 16
MIN, 7
MOLNÁR MIHÁLY, 8; 29

NAGY MEMÓRIA, 12

OR.C, 36
OR.H, 37

PÁRATLAN, 27
PÁROS, 27

T: „VÉGE" OPERÁTOR, 11
TEST.C, 33

ÜTEMEZÉS, 5

VÉGREHAJTÁS, 11

X/Y, 13