Debreceni Egyetem

Informatika Kar

Operációkutatási feladat megoldására készített oktatási célú program fejlesztése Delphi 5 rendszerben, LINDO API segítségével

Témavezető:
Dr Bajalinov Erik
(tudományos főmunkatárs)

Készítette:
Takács Tamás Sándor
(programozó matematikus)

Debrecen
2008

Tartalomjegyzék

Köszönetnyilvánítás

Köszönöm tanáraimnak oktató, nevelő munkáját. Különösen Bajalinov tanár úrnak a türelmet.

Köszönöm munkahelyemen kollégáim együttérzését, biztatását. Különösen Román István igazgató úrnak a támogató hozzáállását.

Végül, de egyáltalán nem utolsó sorban köszönöm családomnak, feleségemnek Jutkának, fiainknak, Bálintnak (17) és Benedeknek (13) hogy elviseltek az elmúlt négy év alatt.

1. Bevezetés

Szakdolgozatom elkészítésével egy olyan program megírását vállaltam, amely az operációkutatásban egyik legelterjedtebb ábrázolási forma – az MPS fájlformátum – feldolgozását teszi lehetővé. Feldolgozás alatt itt elsősorban egy új MPS fájl – adott szerkezetű – előállítását, meglévő MPS fájlok módosítását értem a szükséges I/O eszközök alkalmazásával.

Mivel az elsődleges cél nem egy konkrét operációkutatási módszer programozása volt, így a megszerkesztett MPS fájl megoldását külső gyártó által előállított függvénygyűjtemény alkalmazásával oldottam meg. (Számomra is meglepő módon egyébként – az interneten történő többszöri keresés ellenére sem – olyan programot nem találtam, amely támogatná az MPS fájlok eredeti, IBM szabvány szerinti elkészítését.)

Ezek figyelembevételével célkitűzéseim között az alábbiak szerepeltek:

1) <u>felhasználói szempontból:</u>
   a) legyen egyszerű a program felépítése, kezelése;
   b) a lehetőségekhez képest részletes információt kapjon a felhasználó egy-egy hibáról;
   c) a program „vezesse a felhasználó kezét”: lépésről lépésre haladjon egy konkrét probléma feldolgozásával, a szükséges szabályok betartásával;
2) <u>fejlesztői szempontból:</u>
   a) a grafikus komponensek alkalmazása legyen egyszerű, gyorsan és könnyen kezelhető;
   b) a fejlesztőeszköz rendelkezzen hatékony I/O eszközkészlettel;
   c) legyen könnyen kezelhető a más gyártók által előállított komponensek illesztése;

A 2) pont követelményei alapján a Delphi 5 fejlesztőeszköz mellett döntöttem.

Mivel szakdolgozatomnak nem az operációkutatási módszerek megvalósítása volt a célja, így a feladatok konkrét megoldására a LINDO API 5.0 eszközt választottam. Ennek az eszköznek létezik egy ingyenesen használható verziója, amelyben csak bizonyos méretű és bonyolultságú feladatokat lehet megoldani[1]. Sajnos a komolyabb feladatokhoz már a fizetős

[1] Ezzel a licensszel a lineáris, az egészértékű, és a nem-lineáris problémamegoldás támogatott úgy, hogy a változók száma 300 db, a kényszerfeltételek száma 150 db, az egészértékű változók száma 30 db, míg a nemlineáris változók száma szintén 30 db lehet.

verziót kell használni. (A szakdolgozatom írásakor ennek összege 395$ - 3.995$ között van.) Szakdolgozatomban az ingyenes verziót használtam fel, ami a programom használati értékét nem befolyásolja, hisz az API-ból felhasznált eszközök megegyeznek az ingyenes és a fizetős verziókban. Ebben az eszközben rendelkezésre állnak olyan függvénygyűjtemények – dll-elek (Dynamic Linked Library) formájában -, amelyeket integrálva a program környezetébe megfelelő eredményt szolgáltatnak egy-egy konkrét feladat elvégzésben. (Az eredmények teszteléséről bővebben az *Összefoglalásban* szólok)

Mielőtt azonban részletesen bemutatom a programomat, tekintsük át az operációkutatást általánosságban, az MPS fájl formátumot és a LINDO API 5.0 verzió főbb részeit!

2. Az operációkutatásról

2.1. Mi az operációkutatás?

Ez nem fogalmazható meg olyan egzakt módon, mint például az, hogy mi a geometria. Egy lehetséges definíció: az operációkutatás tudományos, matematikai módszerek alkalmazása az iparban, kereskedelemben, államigazgatásban, honvédelemben olyan összetett problémák megoldására, amelyek nagy rendszerek (munkaerő, gép, nyersanyag, pénzeszközök) irányításában és vezetésében lépnek fel. Az operációkutatási feladatok egyik célja az optimalizálás.

Szakdolgozatomban egy olyan számítógépes program elkészítését tűztem ki célul, amely hatékonyan segíti optimalizálási feladatok összeállítását és azok megoldását.

2.2. Történeti áttekintés

Az optimalizálási feladatok közül elsőnek a lineáris programozás vetődött fel. Legelőször az oroszországi L.V. Kantorovics vetette fel a lineáris programozás problémáját (megoldás nélkül) 1939-ben, munkáját a II. világháború megakadályozta.

1941-ben Hitchcock amerikai matematikus az ún. szállítási problémát vetette fel, amit meg is oldott ún. disztribúciós módszerrel. (Ez egy speciális feladattípus és módszer.)

1947-ben a szintén amerikai G.B. Dantzig és társai az ún. szimplex módszerrel minden lineáris programozási feladat megoldására szóló, általános érvényű megoldást dolgoztak ki.

A gazdasági élet különböző területein hatékonyan alkalmazható módszer nagy számításigényű. Érthető tehát, ha szinte a kezdetektől fogva keresték a számítástechnikai megoldásokat. Az 1960-as évek végétől kezdődően egymás után születtek a nagy számítógépeken több száz, vagy akár több ezer feltételt is kezelni tudó számítógépes programcsomagok.

A lineáris programozásnál a feltételek is, a célfüggvény is lineárisak. A gyakorlat azonban egyre több nem lineáris programozási feladatot vetett fel (pl. kvadratikus, hiperbolikus programozás) Ezekben a feltételek, vagy a célfüggvény nem lineárisak. (Gyakran visszavezetik ezeket lineáris feladat megoldására.) Felvetődött az egész értékű - integer - programozás is, ahol a változók csak egész értékeket vehetnek fel. Ha egy-egy paraméter értéke időben változhat, akkor dinamikus programozásról beszélünk – Bellman -, ha a paraméterek a véletlentől függnek, akkor sztochasztikus programozásról van szó. Ha egy adott feladat célfüggvényében az állandókon kívül változtatható paraméterek is szerepelnek, akkor a programozás parametrikus.

A témakörrel rokon a játékelmélet, ahol a stratégia lehet a biológia, az üzleti élet, a szó szoros értelmében vett játék, vagy akár a háború stratégiája is.

2.3. A lineáris programozás alapfeladatai

Adott feltételek mellett (amelyek lineáris egyenlőtlenségekkel vagy egyenletekkel vannak megadva):

a) valamilyen szempontból optimális (pl. termelési) program összeállítása (lineáris célfüggvény optimalizálása a feltételek által megengedett pontok tartományán);

b) kapacitások kihasználásának elemzése;

c) kapacitások bővítésének elemzése.

Érezhető, hogy ezek a termelés, a gazdasági élet szempontjából fontos feladatok. Ezért terjedhettek el az iparban, a mezőgazdaságban is a lineáris programozási (röviden LP) módszerek.

2.4. Egy számítógépes példa az 1980-as évekből

Ez az egyszerű LP példa egy termelési problémát szemléltet, melyet egy IBM 370/145 számítógépen az LPS/360 program segítségével oldottak meg.

A feladat röviden: kétféle ruhamodell gyártási adatai:

	Megmunkálási idő darabonként (perc/db)			**Haszon db-onként (Ft/db)**	**Termelési érték db-onként (Ft/db)**
	szabás	**varrás**	**hegesztés**		
A modell	3	1	1	60	450
B modell	3	4	0	30	500
Maximális igénybevétel ideje műszakonként (perc/műszak)	420	440	80		

A feladat matematikai modellje a következő:

F1 $3x + 3y \leq 420$

F2 $x + 4y \leq 440$

F3 $x \leq 80$ és $x \geq 0$ és $y \geq 0$

Az MPS formátum ezek alapján (az eredeti szintaktikával, *input*):

```
INPUT
NAME                          OPKUT
*          VALTOZOK           FELTETEL
           X                  F1          3
           Y                  F1          3
           X                  F2          1
           Y                  F2          4
           X                  F3          1
           Y                  F3          0
* CELFUGGVENY
           X                  HASZON      60
           Y                  HASZON      30
* KORLATOK
UB         KORLAT             F1          420
UB         KORLAT             F2          440
UB         KORLAT             F3          80
FR         KORLAT             HASZON      0
ENDATA
```

MOVE

DATA	OPKUT
MAXIMIZE	HASZON
BOUNDS	KORLAT

ENDATA

SUMMARY

Az említett LPS/360 program ebből a bemenetből a következő eredményt szolgáltatta (*output*):

VARIABLE	*ENTRIES TPYE*		*SOLUTION ACTIVITY*	*UPPER BOUND*	*LOWER BOUND*	*CURRENT COST*	*REDUCED COST*
X	*B**	*4*	*80,000*	*******	*0,0*	*60,000*	*0,000*
F1	*UL*	*0*	*420,000*	*420,000*	*0,0*	*0,000*	*-10,000*
Y	*B**	*4*	*60,000*	*******	*0,0*	*30,000*	*0,000*
F2	*B**	*0*	*320,000*	*440,000*	*0,0*	*0,000*	*0,000*
F3	*UL*	*0*	*80,000*	*80,000*	*0,0*	*0,000*	*-30,000*
HASZON	*B**	*0*	*6600,000*	*******	*0,0*	*-1,000*	*-1,000*

Az első oszlop (VARIABLE) mutatja, hogy két változónk van (X és Y), és három feltétel (F1, F2, F3). A SOLUTION ACTIVITY oszlop azt mutatja, hogy az optimális program esetén

X=80 és Y=60,

szabászat kapacitásból 420 perc van kihasználva (F1), a varrógép kapacitásból 320 perc a kihasznált (F2), a hegesztés kapacitásból 80 perc a kihasznált (F3).

Ha ezeket az értékeket összehasonlítjuk a UPPER BOUND oszloppal, látható, hogy a részlegek kihasználtsága a következő:

	SOLUTION ACTIVITY	UPPER BOUND	Kihasználatlan perc/műszak
F1 szabászat	420	420	420 – 420 = 0
F2 varroda	320	440	440 – 320 = 120
F3 hegesztés	80	80	80 – 80 = 0

A REDUCED COST negatív előjellel az árnyékárat mutatja:

	Árnyékár (Ft/perc)
F1 szabászat	10
F2 varroda	0
F3 hegesztés	30

Az inputban a ≤, ≥, = jeleknek megfelelő korlát (BOUNDS) lehetséges értékei:

Felső korlát	UB (UPPER BOUND)	≤
Alsó korlát	LB (LOWER BOUND)	≥
Fix korlát	FX (FIX)	=
Szabad korlát	FR (FREE)	Nincs korlátozó adat

A „Szabad korlát”-ot a célfüggvény bevitelekor adtuk meg (mintha 60x + 30y – H = 0 alakra rendeznénk).

Az MPS formátumú input fájlban tehát a következőket kell megadni:

1. a változók együtthatóit az egyes feltételekben;
2. a célfüggvény együtthatóit;
3. a korlátokat;
4. azt, hogy a feladatot maximalizálni (MAXIMIZE HASZON), vagy minimalizálni (MINIMIZE) kell;

Szakdolgozatomban egy olyan PC-s programot készítettem, amely ennek a formátumú fájlnak (MPS fájlnak) a kezelését – létrehozását, módosítását, megoldását – teszi lehetővé.

A következő részben nézzük meg, milyen változáson ment át a fentebb említett input fájl formátum.

3. Az MPS formátum

3.1. Történet

Az MPS (Mathematical Programming System) formátumot az IBM fejlesztette ki azzal a céllal, hogy egységes tárolási formát vezessen be a matematikai programozási feladatok megadására. Az elgondolás eredetileg lyukkártyás megoldáson alapult: a 80

oszlopos kártyán az előre rögzített hosszúságú mezők a feladat adott részeit tartalmazták a 3.1. ábra szerint[2].

3.2. Leírás

Amint a 3.1. ábrán látható, definíció szerint az alábbi szekciókat tartalmazhatja az MPS fájl:

1. NAME (név)
2. ROWS (sorok)
3. COLUMNS (oszlopok)
4. RHS (right-hand side – jobb oldali feltételek)
5. RANGES (tartományok - opcionális)
6. BOUNDS (korlátok - opcionális)
7. ENDATA (végjel)

Ezeket a szekciókat az alábbi tartalommal kell/lehet feltölteni:

1. **NAME**: (1-4 pozíción) Az adott feladat nevét a 15-22 pozícióban kell megadni. Ennek a szekciónak mindenképpen az első sorban kell lennie. Az ez előtti sorok tartalma nincs értelmezve! (Kötelező szekció)

2. **ROWS**: (1-4 pozíción) Ez a kulcsszó egyedül szerepelhet az adott sorban, és az utána következő szekció a feladatra vonatkozó sorok definícióját – típusát, azonosítóját – tartalmazzák. Sor típusok lehetnek:

E	Egyenlő (equality)	=
L	Kisebb egyenlő (less than or equal)	≤
G	Nagyobb egyenlő (greater than or equal)	≥
N	Célfüggvény (objective)	

[2] Napjainkra a feladatok specializálódása miatt újabb szekciókat vezettek be, melyet részletesebben a LINDO API dokumentációban találhatunk meg.

A sor azonosítója a 15-22 pozícióra kerül, de nem tartalmazhat speciális karaktereket (szóköz, +, -, =, *).

3. **COLUMNS**: (1-7 pozíción) Ez a kulcsszó is egyedül szerepelhet egy sorban. Ez a szekció az adott feladat változóit tartalmazza úgy, hogy az 5-12. pozíción az oszlop azonosítót, a 15-22. pozíción a sor azonosítót, a 25-36. pozíción pedig az adott sor adott együtthatójának értékét tartalmazza. Csak olyan sor azonosítót lehet megadni, amelyet a ROWS szekcióban deklaráltunk! (Kötelező szekció)

4. **RHS**: (1-3 pozíción) Ez a kulcsszó is egyedül szerepelhet egy sorban. Ebben a szekcióban az adott feladat jobboldali értékeit kell megadni a következőképpen: az 5-12. pozíción az adott értékek azonosítóját, a 15-22. pozíción a sor azonosítót, a 25-36. pozíción pedig az adott sor jobboldali értékét tartalmazza. Csak olyan sor azonosító szerepelhet ebben a szekcióban is, amit a ROWS szekcióban megadtunk! A 40-47 és az 50-61 pozíciók kitöltése nem kötelező, egyébként ugyanolyan szabályok vonatkoznak rájuk, mint a 15-22 és a 25-36. pozíciókra. (Kötelező szekció)

5. **RANGES** (opcionális – 1-6 pozíción) Ennek a kulcsszónak is egyedül kell szerepelnie egy sorban. A szekció azokat a sorokat tartalmazza, amelyekre tartomány alakú feltétel(eke)t akarunk megadni. A szekción belül egy sornak a felosztása: az 5-12. pozíción a RANGES vektor azonosítót, a 15-22. pozíción a sor azonosítót, a 25-36. pozíción pedig az adott tartomány alakú értéket tartalmazza. Ennek az értéknek a kialakítása:

$$ha\ L_i \le \sum_{j=1}^{n} a_{ij} x_j \le U_i\ ,\ \ akkor\ R = U_i - L_i$$

és U_i értéke kerül a 25-36. pozícióra. Itt is csak olyan sor azonosító szerepelhet a szekcióban, amit a ROWS szekcióban már megadtunk!

6. **BOUNDS** (opcionális – 1-6 pozíción) Ennek a kulcsszónak is egyedül kell szerepelnie egy sorban. A szekció azokat a sorokat tartalmazza, amelyekre korlátot akarunk megadni. Alapértelmezés szerint minden együttható nem-negatív. Az egyes mezők tartalma lehet:

2-3. pozíció:

LO	Alsó korlát (lower)	$K \leq x_j \leq \infty$
UP	Felső korlát (upper)	$0 \leq x_j \leq K$
FX	Rögzített (fixed)	$x_j = K$
FR	Korlátlan értékű változó (free)	$-\infty \leq x_j \leq \infty$
MI	Nem pozitív	$-\infty \leq x_j \leq 0$
PL	Nem negatív	$0 \leq x_j \leq \infty$

5-12. pozíció:

Ezen a pozíción lévő érték fogja beazonosítani az adott együtthatóra (oszlop) megadott korlátot;

15-22. pozíció:

Annak az oszlopnak az azonosítója, amelyik tartalmazza az adott együtthatót. Csak olyan érték adható meg, amit a COLUMNS szekcióban deklaráltunk;

25-36. pozíció:

Az adott korlát értéke.

7. **ENDATA:** Ez a kulcsszó is csak egyszer szerepelhet egy lyukkártyán, az adott feladat végét jelöli. Az utána következő sorokba írt adatok nincsenek értelmezve. (Kötelező szekció)

3.3. Fájl formátum

Napjainkra az MPS lyukkártya tárolási módját felváltotta a szöveges fájlformátum. Ezek szerint hagyományos – szekvenciális - text formátumban írható/olvasható egy adott MPS feladat. A fájlformátum gyakorlatilag követi a lyukkártyás megjelenést: legalább 5 db. szekcióból – sorból – és 6 db. oszlopból áll. Ennek megfelelően mátrix alakban reprezentálható formátumot kapunk. A *2.4. fejezet* példája a mai alap MPS formátummal a *3.3.1. ábrán* látható

4. LINDO API

4.1. Mi a LINDO API?

A LINDO API (Application Programming Interface) egy olyan eszközkészlet, melynek segítségével optimalizálási feladatokat lehet megoldani. Könnyen applikálható formában tartalmazza azokat az eljárásokat, amelyeket az elmúlt évtizedekben fejlesztettek ki az ilyen jellegű problémák megoldására. A készítők egyszerűségre való törekvését jól jelzi, hogy a LINDO API része egy parancssorból futtatható program, amely megfelelően paraméterezve ugyancsak képes megoldani a lentebb említett feladatokat. A fejlesztőeszköz elérhető Windows és Unix platformokon egyaránt.

Szakdolgozatomban a LINDO internetes oldaláról ingyenesen letölthető LINDO API 5.0 csomag tartalmát használtam fel. Mivel dolgozatom tárgya nem egy konkrét megoldási módszer alkalmazása, így kézenfekvő megoldásnak kínálkozott egy meglévő eszközkészlet felhasználása[3].

4.2. A LINDO API integrált módszerei

A LINDO API-t az alábbi problémák megoldására fejlesztették ki:

1. **lineáris programozási** (linear programming - LP) feladatokra az alábbi módszerek alkalmazásával:
    - primal simplex;
    - dual simplex;
    - Barrier módszer;
2. **vegyes-egészértékű** (mixed-integer) feladatokra a szétválasztás-és-vágás (branch-and-cut) módszer alkalmazásával
3. **nem lineáris** feladatokra az alábbi módszerek alkalmazásával:
    - egymásra épülő lineáris módszer (successive linear programming – SLP);
    - általánosított csökkentett gradiens módszer (generalized reduced gradient – GRG);
4. **általános** programozási feladatok (global solver) megoldására, melyet visszavezetnek nem lineáris feladat megoldására.

[3] Az internetes oldal pontos elérhetőségét lásd az *Irodalomjegyzékben*.

4.3. Feladatmegoldás LINDO API segítségével

4.3.1. Általános módszer

A LINDO API-ba integrált programozási környezetben az alábbi lehetőségek állnak rendelkezésre egy feladat megoldására:

a) a modellt jellemző adatstruktúrától (tömb reprezentáció) a LINDO API-ba integrált logikai adatmodellen keresztül a rendelkezésre álló rutinok használata;

b) a modell beolvasása fájlból közvetlenül a LINDO API-ba a rendelkezésre álló input/output eljárásokon keresztül;

Az alábbi egyszerű példa az a) esetre koncentrál, vagyis bemutatja, hogy a tömb reprezentáció hogyan jellemzi az LP feladatot egy adott programozási környezetben.
Legyen adott négy együtthatóval (n=4) és négy kényszerfeltétellel (m=4) az alábbi feladat:

Célfüggvény:

$x_1 + x_2 + x_3 + x_4 \rightarrow \min$

Kényszerfeltételek:

$$
\begin{array}{llllllll}
3x_1 & & & & & + & 2x_4 & = & 20 \\
 & & 6x_2 & & & + & 9x_4 & \geq & 20 \\
4x_1 & + & 5x_2 & + & 8x_3 & & & = & 40 \\
 & & 7x_2 & & 1x_3 & & & \geq & 10
\end{array}
$$

Az együtthatók **alsó és felső korlátja** egyértelműen definiált (ha egyik korlát sincs megadva, feltételezzük, hogy a változók értékei folytonosak, alsó korlátjuk nulla, felső korlátjuk pedig végtelen):

$$
\begin{array}{lllll}
2 & \leq & x_1 & \leq & 5 \\
1 & \leq & x_2 & \leq & +\infty \\
-\infty & \leq & x_3 & \leq & 10 \\
-\infty & \leq & x_4 & \leq & +\infty
\end{array}
$$

Az *Ábrajegyzék* 4.1. ábrája azt mutatja meg, hogy a mátrix kényszerfeltételeinek (sorainak) együtthatóit kiemelve triviális a változók tömbben való ábrázolása. Az ábrán **A, B, C, D**, és **E** elnevezésekkel rendre a célfüggvényt, a változókkal végzett műveleteket, a jobboldali értékeket az alsó- és a felső korlátokat jelöltem be: (A **B** tömb elemei az eredeti MPS szabványnak megfelelően a 2. fejezet *ROWS* szekciójában leírt betűjelek lehetnek.)

Ebből kiindulva az alábbi tömböket állíthatjuk elő:

A = [1 1 1 1]

B = [E G E G]

C = [20 20 40 10]

D = [2 1 –LS_INFINITY –LS_INFINITY]

E = [5 LS_INFINITY 10 LS_INFINITY]

(A **D** és **E** tömbökben szereplő LS_INFINITY a LINDO API konstans értékei, a végtelen ábrázolására szolgálnak.)

Az így kialakított tömb reprezentációban minden tömbelem megfelelő típusú, amit már a LINDO API beépített eljárásaival fel lehet dolgozni. Általában a kényszerfeltételek változóit egy egydimenziós tömbben tároljuk, de különböző reprezentációkban az ún. *ritka mátrixot* alkalmazzuk[4].

4.3.2. Egy konkrét példa

A *4.3.1. fejezetben* már említett példán keresztül mutatom be, hogyan kell integrálni egy tetszőleges fejlesztőrendszerbe a LINDO API eszközöket.

Még egyszer a példa:

A célfüggvény:

$x_1 + x_2 + x_3 + x_4 \rightarrow \min$

A kényszerfeltételek:

$$
\begin{array}{llllllll}
3x_1 & & & & & + & 2x_4 & = & 20 \\
 & & 6x_2 & & & + & 9x_4 & \geq & 20 \\
4x_1 & + & 5x_2 & + & 8x_3 & & & = & 40 \\
 & & 7x_2 & & 1x_3 & & & \geq & 10
\end{array}
$$

[4] Részletesebben lásd a LINDO API dokumentációt.

A megszorítások:

$2 \leq x_1 \leq 5$

$1 \leq x_2 \leq +\infty$

$-\infty \leq x_3 \leq 10$

$-\infty \leq x_4 \leq +\infty$

Első lépés a LINDO API használatakor, hogy inicializálnunk kell a szükséges LINDO környezetet. Ebben megállapításra kerül a felhasznált licensz – ez befolyásolja a megoldható feladat bonyolultságát és nagyságát (lásd még a *Bevezetőben* leírtakat) -, valamint meghatározásra kerül egy azonosító - handles -, amely a szükséges erőforrásokat fogja azonosítani (pl egy mutatót a felhasznált memória területre). Ez az objektum a LINDO környezet deklarálásával jön létre. Ennek reprezentációja az alábbi kódrészletben látható (a forráskód eredeti angol szövegét megtartottam, csak a kommenteket fordítottam le):

```
/* LINDO API környezeti változójának deklarálása */
pLSenv pEnv;
/* az operáció kutatási modell változójának deklarálása */
pLSmodel pModel;
/* LINDO API környezet létrehozása. A MY_LICENSE_KEY beépített LINDO makró
hivatkozása */
pEnv = LScreateEnv ( &nErrorCode, MY_LICENSE_KEY);
/* az operáció kutatási modell létrehozása */
pModel = LScreateModel ( pEnv, &nErrorCode);
```

A következő lépésekben az LP adatok kerülnek meghatározásra és betöltésre a LINDO API-ba.

1. Ehhez először a célfüggvény irányát kell meghatározni, ami ugye lehet maximalizálás vagy minimalizálás (LS_MAX, LS_MIN).

A példában szereplő célfüggvényt minimalizálni kell, a konstansok ábrázolása lebegőpontos számformátummal történik, és végül a kényszerfeltételből négy darab van, amit egy tömbben fogunk tárolni, tehát rendre a következő sorok kerülnek a forráskódba (a tömbelemek értékei értelemszerűen a célfüggvény konstans értékei, jelen esetben mind a négy 1):

int nDir = PS_MIN; / A célfüggvényt minimalizálni kell */*
double dObjConst = 0.0; / Célfüggvény értéke */*
double adC[4] = [1., 1., 1., 1.]; / A célfüggvény együtthatóinak értéke */*

2. A második lépésben a kényszerfeltételeket kell deklarálnunk, nevezetesen a kényszerfeltételek – a feladat sorainak – számát, a jobb oldali értékeket ábrázoló tömböt, a kényszerfeltételek műveleteit ábrázoló tömböt (lásd *3.2. fejezet* ROWS szekció leírását), végül a kényszerfeltételekben szereplő nem-nulla értékű együtthatók ábrázolására szintén egy tömböt kell megadnunk. A forráskódba tehát rendre a következő sorok kerülnek:

int nM = 4; / kényszerfeltételek száma */*
double adB[4] = [20., 20., 40., 10.]; / jobb oldali értékek tömbje */*
char acConTypes[4] = ['E', 'G', 'E', 'G']; / kényszerfeltételek műveleteinek tömbje */*
int nZ = 9; / nem-nulla értékű változók száma */*

A kényszerfeltételeket leíró mátrixban minden egyes oszlopnak meg kell adnunk a hosszát, amelyet egy egész típusú mutatóval tudunk megtenni – a példában ennek kezdőértékét NULL-ra kell állítanunk -, majd a *ritka mátrix* reprezentációnak megfelelően meg kell adnunk azokat a tömböket, amelyek beazonosítják a kényszerfeltétel megfelelő együtthatóit. Ezek alapján a következő sorok kerülnek a forráskódba:

*int *pnLenCol = NULL; /* oszlopok hossza */*
int anBegCol[5] = { 0 , 2 , 5 , 7 , 9 }; / azon oszlopok azonosítója, amelyekben nem-nulla értékű változók vannak */*
double adA[9] = { 3.0, 4.0, 6.0, 5.0, 7.0, 8.0, 1.0, 2.0, 9.0 }; / a nem-nulla értékű együtthatók értékei */*
int anRowX[9] = { 0 , 2 , 1 , 2 , 3 , 2 , 3 , 0 , 1 }; / azon sorok azonosítója, amelyekben a nem-nulla értékű együtthatók találhatóak */*

3. A harmadik lépésben a változókat kell deklarálnunk. Meg kell adnunk a darabszámukat, az esetleges megkötésük értékeit tömbök formájában (alsó-, ill. felső korlátok), végül a típusaikat meghatározó tömböt kell megadnunk. A forráskódba tehát rendre a következő sorok kerülnek:

```
int nN = 4; /* változók száma */
double pdLower[4] = {2, 1, -LS_INFINITY, -LS_INFINITY}; /* alsó korlátok értékei */
double pdUpper[4] = {5, LS_INFINITY, 10, LS_INFINITY}; /* felső korlátok értékei */
char acVarTypes[4] = {'C','C','C','C'}; /* változók típusa */
```

Ha megadtuk a feladatot, ezeket az információkat – programváltozókat – át kell adnunk a LINDO API-nak a következő formában:

```
nErrorCode = LSloadLPData( pModel, nM, nN, nDir, dObjConst, adC, adB,
acConTypes, nNZ, anBegCol, pnLenCol, adA, anRowX, pdLower, pdUpper);
```

Minden LINDO API függvény egy hibakódot – nErrorCode – ad vissza eredményül arról, hogy az adott függvény sikeresen lefutott-e. Ha a hibakód értéke 0, minden rendben van, ellenkező esetben a LINDO API-hoz adott dokumentációból kikereshető – vagy a programunk által szöveges formában is kezelhető – hibaüzenetet kapunk. A hibás futás eredményét programunkban kivételkezeléssel tudjuk orvosolni.

4. A negyedik lépésben a megoldás maradt hátra. Ezt a LINDO API-ban, ha egyszerű LP feladatról van szó, a primál szimplex módszer alkalmazásával tehetjük meg, ellenkező esetben rendelkezésre áll még a duál szimplex, illetve a Barrier módszer is (ha van rá licenszünk) – lásd még a 4.2. fejezetet. A megoldáshoz a következő sort kell beillesztenünk a forráskódunkba:

```
nErrorCode = LSoptimize( pModel, LS_METHOD_PSIMPLEX, &nSolStat); /* a feladat
megoldása */
```

5. Az utolsó előtti lépésben az eredmény kiíratása történhet meg. Lehetőségünk van a célfüggvény eredményének, valamint az egyes változók értékeinek kiíratására, amihez egy újabb lebegőpontos típusú változókat tartalmazó tömböt kell deklarálnunk. A forráskódba a következő sorok kerülnek (a kiíratás ebben a példában C-nyelv típusú):

```
nErrorCode = LSgetInfo( pModel, LS_DINFO_POBJ, &dObj); /* célfüggvény értékének
meghatározása */
printf( "Objective Value = %g\n", dObj); /* a célfüggvény értékének kiírása */
```

```
double adX[4]; /* tömb deklarálása a változók értékeinek meghatározásához */
nErrorCode = LSgetPrimalSolution ( pModel, adX); /* változók értékeinek meghatározása */
/* változók értékeinek kiíratása */
printf ("Primal values \n");
for (i = 0; i < nN; i++) printf( " x[%d] = %g\n", i, adX[i]);
printf ("\n");
```

6. Végül az utolsó lépésben fel kell szabadítani a LINDO API környezet számára lefoglalt memóriaterületet, amit a következő sor beszúrásával tehetünk meg:

```
nErrorCode = LSdeleteEnv( &pEnv);
```

7. A megoldás eredménye:

```
Objective Value = 10.44118
Primal values
x[0] = 5
x[1] = 1.176471
x[2] = 1.764706
x[3] = 2.5
```

8. A példafeladat MPS formátuma így néz ki (az általam készített HDI programmal)

```
NAME  proba
ROWS
E  R1
G  R2
E  R3
G  R4
N  R5
COLUMNS
 x1 R1 3.0000 R3 4.0000
 x1 R5 1.0000
 x2 R2 6.0000 R3 5.0000
 x2 R4 7.0000 R5 1.0000
 x3 R3 8.0000 R4 1.0000
 x3 R5 1.0000
 x4 R1 2.0000 R2 9.0000
 x4 R5 1.0000
RHS
 rhs1 R1 20.0000
 rhs1 R2 20.0000
 rhs1 R3 40.0000
 rhs1 R4 10.0000
RANGES

BOUNDS
LO BND1 x1 2.0000
UP BND1 x1 5.0000
LO BND1 x2 1.0000
PL BND1 x2
MI BND1 x3
UP BND1 x3 10.0000
FR BND1 x4
ENDATA
```

4.3.3. Fontosabb függvények

A LINDO API 5.0 verziójában közel 150 db függvény áll rendelkezésre egy-egy konkrét operációkutatási feladat megoldására. Ezek között vannak – többek között - memóriakezelő, matematikai, és input/output kezelő eljárások[5] egyaránt.

Szakdolgozatomban én csak az általam felhasznált függvényeket részletezem, a LINDO API besorolásuk alapján.

Licensz- és verzió információkat szolgáltató függvények.

LSloadLicenseString()

Leírás.

A paramétereként kapott fájlból text formátumban kiolvassa a rendelkezésre álló licensz információkat. (Amint azt a bevezetőben írtam, a LINDO API-nak létezik fizetős

[5] A teljes API dokumentáció hozzáférhető már az általam használt verzióban is, melynek terjedelme 512 oldal.

verziója, amely bonyolultabb feladatok megoldását is lehetővé teszi. Annak eldöntése, hogy milyen feladatokat lehet megoldatni az API függvényekkel, a licensz információk alapján történik.)

Visszatérési érték.

0, ha rendben van a licensz, minden más esetben egy hibakód, melyet a LINDO API dokumentációja részletesen tartalmaz.

Prototípus.

int	LSloadLicenseString(char *pszFname, char *pszLicense)

Input paraméterek.

Név	**Jelentés**
pszFname	Egy *null* végjelű sztringre mutató változó, amely a licensz információkat tartalmazó fájl nevét hivatkozza. Ennek a fájnak a neve általában *lndapi50.lic.*

Output paraméterek.

Név	**Jelentés**
pszLicensee	Egy *null* végjelű sztringre mutató változó, amely a licensz információkat tartalmazza.

Struktúra létrehozó és törlő függvények

LScreateEnv()

Leírás.

Létrehoz egy új *LSenv* példányt, amelyben majd használni lehet a legfontosabb modelleket. Az *LSenv* struktúra az API-val szállított *lindo.h* fájlban található.

Visszatérési érték.

Ha sikerült létrehozni az új példányt, akkor a példány memória területét hivatkozó mutató, egyébként NULL.

Prototípus.

pLSenv	LScreateEnv(int *pnErrorcode, char *pszPassword)

Input paraméterek.

Név	**Jelentés**
pszPassword	A licensz kulcsot tartalmazó sztringet hivatkozó mutató.

Output paraméterek.

Név	**Jelentés**
pnErrorcode	A hibakódot hivatkozó mutató. Ha nincs hiba, értéke 0.

LScreateModel()

Leírás.

Létrehoz egy új *LSmodel* példányt.

Visszatérési érték.

Ha sikerült létrehozni az új példányt, akkor a példány memória területét hivatkozó mutató, egyébként NULL.

Prototípus.

pLSmodel	LScreateModel(pLSenv pEnv, int *pnErrorcode)

Input paraméterek.

Név	**Jelentés**
pEnv	A modellt tartalmazó környezet hivatkozása. (Lásd *LScreateEnv()* függvény)

Output paraméterek.

Név	Jelentés
pnErrorcode	A hibakódot hivatkozó mutató. Ha nincs hiba, értéke 0.

Megjegyzés: ennek a függvénynek a meghívását meg kell előznie az *LScreateEnv()* függvény meghívásának.

LSdeleteEnv()

Leírás.

LSenv példány törlése. A példány által használt memória területet felszabadítja, az adott tárterületet hivatkozó mutató értékét NULL-ra állítja.

Visszatérési érték.

Ha sikerült letörölni az adott példányt, akkor 0, egyébként pedig a LINDO API dokumentációjában megtalálható hibakód.

Prototípus.

int	LSdeleteEnv(pLSenv *pEnv)

Input paraméterek.

Név	Jelentés
pEnv	A törlendő példányt tartalmazó környezet hivatkozása.

INPUT/OUTPUT függvények.

LSreadMPSFile()

Leírás.

MPS formátumú fájlból beolvassa az adatokat, és betölti azokat egy megadott probléma-struktúrába.

Visszatérési érték.

Ha a beolvasás sikerült, 0, egyébként a LINDO API dokumentációjában megtalálható hibakód.

Prototípus.

int	LSreadMPSFile(pLSmodel pModel, char *pszFname, int nFormat)

Paraméterek.

Név	**Jelentés**
pModel	Az adott modell memóriabeli területét hivatkozó mutató. (Lásd az *LScreateModel()* függvényt)
pszFname	Az MPS fájl nevét – elérési úttal – tartalmazó, *null* végjelű sztringet hivatkozó mutató.
nFormat	Egy egész típusú konstans, amely meghatározza, hogy a beolvasandó MPS fájl formázott-e. Az API-ba épített makró hivatkozásoknak megfelelően értéke lehet LS_FORMATTED_MPS vagy LS_UNFORMATTED_MPS

Megjegyzés: Ha egy MPS fájlt az LS_FORMATTED_MPS paraméterrel olvasunk be, akkor minden oszlop szóközöket tartalmazó literálként lesz beolvasva, ahol minden változó névnek 8 karakterből kell állnia. Ellenkező esetben a fájl tartalmazhat nem csak szóköz elválasztókat – pl. tabulátor jeleket -, és egy-egy sor hossza lehet 256 karakter is.

LSreadLINDOFile()

Leírás.

LINDO formátumú fájlból beolvassa az adatokat, és betölti azokat egy megadott probléma-struktúrába.

Visszatérési érték.

Ha a beolvasás sikerült, 0, egyébként a LINDO API dokumentációjában megtalálható hibakód.

Prototípus.

int	LSreadLINDOFile(pLSmodel pModel, char *pszFname)

Paraméterek.

Név	**Jelentés**
pModel	Az adott modell memóriabeli területét hivatkozó mutató. (Lásd az *LScreateModel()* függvényt)
pszFname	Az MPS fájl nevét – elérési úttal – tartalmazó, *null* végjelű sztringet hivatkozó mutató.

LSreadMPIFile()

Leírás.

MPI formátumú fájlból beolvassa az adatokat, és betölti azokat egy megadott probléma-struktúrába.

Visszatérési érték.

Ha a beolvasás sikerült, 0, egyébként a LINDO API dokumentációjában megtalálható hibakód.

Prototípus.

int	LSreadMPIFile(pLSmodel pModel, char *pszFname)

Paraméterek.

Név	**Jelentés**
pModel	Az adott modell memóriabeli területét hivatkozó mutató. (Lásd az *LScreateModel()* függvényt)
pszFname	Az MPS fájl nevét – elérési úttal – tartalmazó, *null* végjelű sztringet hivatkozó mutató.

Megoldás lekérdező függvények

LSgetInfo()

Leírás.

Miután a LINDO API függvényeivel megoldásra került egy adott modell, ezzel a függvénnyel a modellről és/vagy a megoldás eredményeiről kaphatunk információkat. A függvény nem alkalmazható visszamenőleges információk megszerzésére.

Visszatérési érték.

Ha az információ megszerzése sikerült, 0, egyébként a LINDO API dokumentációjában megtalálható hibakód.

Prototípus.

int	LSgetInfo(pLSmodel pModel, int nQuery, void *pvValue)

Input paraméterek.

Név	**Jelentés**
pModel	Az adott modell memóriabeli területét hivatkozó mutató. (Lásd az *LScreateModel()* függvényt)
nQuery	A lekérdezés típusát megadó egész típusú érték. A beépített LINDO makrók alapján a programomban az alábbiakat használtam fel[6]: • LS_IINFO_NUM_VARS: az adott feladatban szereplő változók számát adja eredményül • LS_IINFO_NUM_CONS: az adott feladatban szereplő együtthatók számát adja eredményül • LS_IINFO_NUM_NONZ: „nem-nulla” értékű változók számát adja eredményül • LS_IINFO_NUM_CONT: „folyamatos értékű” változók számát adja eredményül • LS_DINFO_MIP_OBJ: MIP modell célfüggvény értékét adja

[6] A teljes makró lista megtalálható a LINDO API dokumentációban (összesen 157 db van).

	eredményül • LS_DINFO_POBJ: Primál feladat célfüggvény értékét adja eredményül

Output paraméterek.

Név	Jelentés
pvValue	A keresett információ memória területét hivatkozó mutató. A függvény meghívása előtt le kell foglalni a szükséges memória méretet.

LSgetPrimalSolution()

Leírás.

Az adott modell primál változóit adja vissza

Visszatérési érték.

Ha minden rendben, 0, egyébként a LINDO API dokumentációjában megtalálható hibakód.

Prototípus.

int	LSgetPrimalSolution(pLSmodel pModel, double *padPrimal)

Input paraméter.

Név	Jelentés
pModel	Az adott modell memóriabeli területét hivatkozó mutató. (Lásd az *LScreateModel()* függvényt)

Output paraméter.

Név	Jelentés
padPrimal	Annak a tömbnek a hivatkozása, amelyikbe a változók értékei kerülnek. A tömbnek legalább akkorának kell lennie, mint amennyi primál változó az adott feladatban szerepel. A tömbelemek típusa lebegő pontos számformátum.

LSgetMIPPrimalSolution()

Leírás.

Az adott feladat primál változóit adja vissza az aktuális MIP modellben.

Visszatérési érték.

Ha minden rendben, 0, egyébként a LINDO API dokumentációjában megtalálható hibakód.

Prototípus.

int	LSgetMIPPrimalSolution(pLSmodel pModel, double *padPrimal)

Input paraméter.

Név	**Jelentés**
pModel	Az adott modell memóriabeli területét hivatkozó mutató. (Lásd az *LScreateModel()* függvényt)

Output paraméter.

Név	**Jelentés**
padPrimal	Annak a tömbnek a hivatkozása, amelyikbe a változók értékei kerülnek. A tömbnek legalább akkorának kell lennie, mint amennyi primál változó az adott feladatban szerepel. A tömbelemek típusa lebegő pontos számformátum.

Visszatérési értéket beállító függvények.

LSsetCallback()

Leírás.

A LINDO API-val szállított visszatérési értéke(ke)t beállító függvények a LINDO API futásának teljes időtartama alatt bármikor használhatóak. A felhasználók által megírt ilyen jellegű függvényekkel pedig befolyásolni tudjuk egy-egy megoldás futását (pl. szükség esetén megszakíthatjuk azt).

Visszatérési érték.

Ha minden rendben, 0, egyébként a LINDO API dokumentációjában megtalálható hibakód.

Prototípus.

int	LSsetCallback(pLSmodel pModel, int (CALLBACKTYPE *pcbFunc)(LSmodel*, int, void*), void *pvData)

Paraméterek.

Név	**Jelentés**
pModel	Az adott modell memóriabeli területét hivatkozó mutató. (Lásd az *LScreateModel()* függvényt)
pcbFunc	Mutató a felhasználó által írt visszatérési értéket beállító függvényre. (Deplhi-ben pl. makró segítségével egy lehetséges megoldás: *cbf := MyCallback; @cbf;*)

Megjegyzések:

- ha nincs felhasználó által írt függvény, a *pcbFunc* paraméter értéke legyen NULL;
- ha a *pcbFunc* függvény visszatérési értéke nem nulla, a LINDO API megszakítja a megoldás végrehajtását, amit a programozónak kell lekezelnie;

LSgetCallbackInfo()

Leírás.

A LINDO API optimalizálási eljárás alkalmazása során, a megoldás egy adott pillanatában információk lekérdezése a feladatról.

Visszatérési érték.

Ha minden rendben, 0, egyébként a LINDO API dokumentációjában megtalálható hibakód.

Prototípus.

int	LSgetCallbackInfo(pLSmodel pModel, int nLocation, int nQuery, void *pvValue)

Input paraméterek.

Név	**Jelentés**
pModel	Az adott modell memóriabeli területét hivatkozó mutató. (Lásd az *LScreateModel()* függvényt)
nLocation	A megoldás „helye”, annak a programozói függvénynek a neve, amelyikben a megoldás „történik”.
nQuery	A LINDO API-tól várt információk meghatározására szolgáló egész típusú paraméter. A programomban az alábbiakat használtam fel[7] • LS_IINFO_SIM_ITER: iterációk száma a szimplex módszerrel végrehajtott feladatmegoldás során • LS_DINFO_POBJ: primál feladat célfüggvényének értékét adja eredményül: • LS_IINFO_MIP_SIM_ITER: iterációk száma a szimplex módszerrel végrehajtott a MIP feladatmegoldás során • LS_DINFO_MIP_OBJ: MIP feladat célfüggvény értéke • LS_IINFO_MIP_STATUS: a MIP feladat végrehajtásának aktuális állapotát adja eredményül

Optimalizálási függvények.

LSoptimize()

Leírás.

Egy feladat optimalizálása a megadott módszerrel

[7] A teljes makró lista megtalálható a LINDO API kézikönyvében (47 db van belőle)

Visszatérési érték.

Ha minden rendben, 0, egyébként a LINDO API dokumentációjában megtalálható hibakód.

Prototípus.

int	LSoptimize(pLSmodel pModel, int nMethod, int *pnStatus)

Input paraméterek.

Név	**Jelentés**
pModel	Az adott modell memóriabeli területét hivatkozó mutató. (Lásd az *LScreateModel()* függvényt)
pMethodc	Az optimalizácis eljárást meghatározó egész típusú paraméter. Értékei lehetnek: • LS_METHOD_FREE: 0 • LS_METHOD_PSIMPLEX: 1 • LS_METHOD_DSIMPLEX: 2 • LS_METHOD_BARRIER: 3 • LS_METHOD_NLP: 4 Az LS_METHOD_FREE paraméter alkalmazásával a LINDO API az adott feladatnak leginkább megfelelő optimalizálási eljárást fogja alkalmazni,

LSsolveMIP()

Leírás.

A branch-and-cut eljárást alkalmazó optimalizálási modell.

Visszatérési érték.

Ha minden rendben, 0, egyébként a LINDO API dokumentációjában megtalálható hibakód.

Prototípus.

int	LSsolveMIP(pLSmodel pModel, int *pnStatus)

Input paraméterek.

Név	Jelentés
pModel	Az adott modell memóriabeli területét hivatkozó mutató. (Lásd az *LScreateModel()* függvényt)

Output paraméterek.

Név	Jelentés
pnStatus	Az optimalizálás állapotát leíró egész értékű paramétert hivatkozó mutató.

5. A HDI program (1.0 verzió)

Ebben a fejezetben részletesen ismertetem az előző fejezetekben vázolt problémákat és megoldásaikat megvalósító programomat. Mint a bevezetőben említettem, a Delphi 5 verzióját választottam fejlesztőeszköznek, így a további fejezetekben látható konkrét programkód részletek is ezen a nyelven íródtak. Az egyes ábrákat az SSADM Case eszközeivel készítettem, az *Ábrajegyzékben* szereplő képernyőket az elkészített programból ollóztam ki.

Szakdolgozatom fizikai mellékletét képezi egy CD, rajta a telepíthető HDI programmal – részletesen lásd az *5.5.1. fejezetet*.

Programom elkészítésekor először az jutott eszembe, hogy „hogy csináljam” ezt meg? Mivel párhuzamot találtam egy konkrét operációkutatási feladat megoldása és a program elkészítése között – „hogy kezdjek hozzá”, „hogy csináljam” -, úgy gondoltam, megfelelő fantázianév lesz a „**H**ow **D**o **I**t?” angol kifejezésből összeállított mozaikszó a programom számára. (Megjegyzem a magyar fordítás itt nem pontos, mert szó szerint a „Hogy csináld?” lenne a helyes kifejezés.)

5.1. Célok, lehatárolások

Az MPS fájlok kezelését megvalósító programnak gyakorlatilag egyszerű céljai vannak:

- tudjon vele a felhasználó MPS formátumú **új állományt megszerkeszteni**, fizikai háttértárolóra **elmenteni**;

- **meglévő** – korábban készített – **MPS formátumú fájlt megnyitni**, annak adattartalmát **módosítani**;
- az így elkészített **MPS fájlokat** tudja a program **megoldani** (operációkutatási feladatokról van szó!);
- tudja **kinyomtatni** a fájlt és a megoldást is;
- rendelkezzen hatékony **súgó eszközökkel** az oktatási célok eléréséhez;

Összefoglalva: ha egy olyan felhasználó ül le a program elé, aki már jártas az operációkutatási feladatokban – tisztában van a fogalmakkal -, de még nem ismeri részletesen az MPS fájl formátumot, el tudjon boldogulni a programmal (képes legyen egy-egy konkrét feladat megfogalmazására, megoldására).

Mivel az MPS fájlok az operációkutatási feladatokhoz kötődnek, kézenfekvő lenne, hogy a program ilyen módszerekkel oldja meg a kitűzött feladatokat. Mivel azonban már létezik egy igen hatékony, és a legelterjedtebb magasabb szintű programozási nyelvekhez könnyen integrálható eszközrendszer - a LINDO cég API-ja -, így **a programomnak nem része az egyes operációkutatási módszerek megvalósítása**. Annál érdekesebb kérdés viszont az említett külső gyártó által dll-ek – Dinamyc Linked Library – formájában szállított függvények applikálása, pl. a Delphi-be. Erről részletesen a *4. fejezet* szól, itt már a konkrét folyamatábrákat és a forráskódokat mutatom be.

5.2. Fontosabb szerkezeti diagrammok és forráskódok

Ebben a fejezetben néhány folyamat ábráját mutatom be, az ábrák után a program kódjával kiegészítve a leírást. A megvalósult képernyőket az *Ábrajegyzékben* helyeztem el, a megfelelő helyen hivatkozom rájuk.

5.2.1. A program indítása

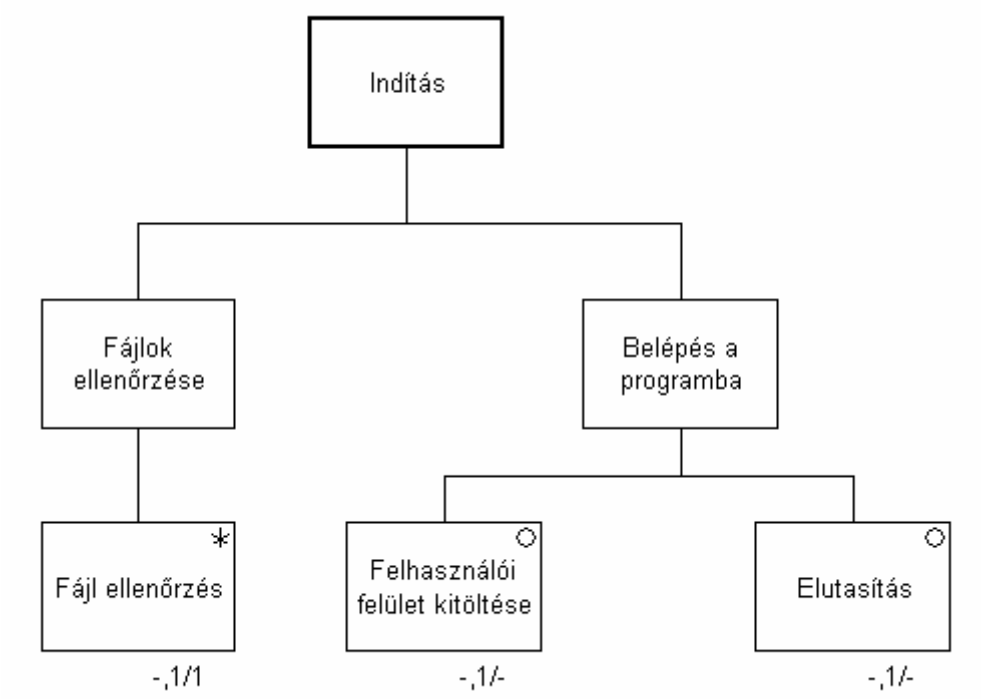
Mielőtt az érdemi munka elkezdődhet a program segítségével, szükséges néhány környezeti körülmény meglétének vizsgálata:

- létezik-e a „hdi.dat” nevű állomány, melyben a futtatás néhány paramétere van beállítva (részletesen lásd az *5.5.1. fejezetet*)
- létezik-e a nyelvi beállításoknak megfelelő állomány (részletesen lásd az *5.5.1. fejezetet*)
- kell-e indítani a logo képernyőt

- nyilván kell tartani az egyszerre megnyitott MPS fájlokat, hogy elkerüljük a módosításból és az újbóli megnyitásból adódó adatvesztés(eke)t

A program tervezése során úgy gondoltam, hogy növeli a program értékét a nyelvválasztás lehetőségének megteremtése. Ezt úgy valósítottam meg, hogy minden, a képernyőn megjelenő információt egy szöveges fájlból olvas ki a program. Ebbe beletartoznak a hibaüzenetek, a menüpontok, az ún. „hint"-ek, és egyéb, az űrlapokon megjelenő szövegek. Ezt a célt szolgálja minden fordítási egységben megjelenő „setCaption" metódus, ami a hibaüzenetek kivételével lekezeli az összes szöveges objektum értékének beállítását. Ezzel sikerült elérnem, hogy nyelvválasztáskor csak egy metódust kelljen meghívnom, illetve hogy egy-egy szövegrész esetleges megváltozása miatt ne kelljen újrafordítani az egész programot. Idő hiányában a magyar és az angol nyelv választható, de minden probléma nélkül gyakorlatilag bármilyen nyelven hozzáférhetővé tehető a program.

A folyamatábra:



A programkód:

```
procedure TMainForm.FormCreate(Sender: TObject);
begin
  //Egyszerre megnyitott fájlok nyilvántartására:
  numOfOpenedFile := 1;
  SetLength(openedMPS, 1);
  openedMPS[0] := '';

  //A program elérési útjának meghatározása:
  GetDir(0, eleresiUt);

  //Kötelező fájlok meglétének vizsgálata:
  if not FileExists(eleresiUt + '\hdi.dat') then
    HandleException(0, eleresiUt);
  if not FileExists(eleresiUt + '\Lang\hdi_msg_hu.dat') then
    HandleException(0, eleresiUt);
  setCaption;

  //Ha kell, logo képernyő indítása:
  if iniFile.ReadInteger('Options', 'Presentation', 0) = 1 then
    Bemutato(eleresiUt);

  //Ha indításkor volt paramétere a programnak, betöltés:
  if FileExists(ParamStr(1)) then CreateMDIChild(ParamStr(1));
end;
```

5.2.2. Egyszerre több megnyitott MPS fájl kezelése

Programomban – az általános elvárásoknak megfelelően – meg kellett oldanom, hogy egyszerre több MPS fájlt lehessen szerkeszteni, akár újat – üreset -, akár egy módosításra megnyitottat. A Delphi 5-ben szerencsére – ennek a problémának a megoldására is – létezik egy igen hatékony eszköz, az ún. MDI (Multi Document Interface) alkalmazás. Az ilyen alkalmazás felépítése Windows környezetben az alábbi:

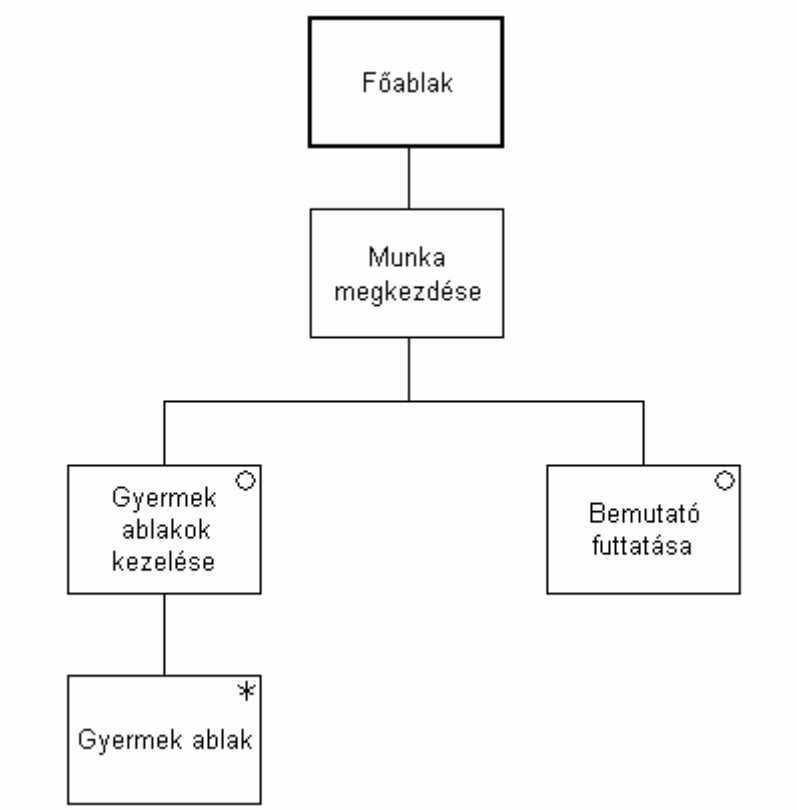
- az alkalmazás főablaka keretként, vagy tárolóként viselkedik (ehhez az ablaknak saját menüszerkezetre és egyedi kódra van szüksége, lásd például az előző fejezet programkódjában a „TMainForm” minősített hivatkozást a „FormCreate” eljárás deklaráló sorában);
- a MDI alkalmazásoknak több gyermekablaka is lehet, amelyek azonos vagy különböző típusúak lehetnek. Ezek a gyermekablakok nem közvetlenül a keretablakba vannak helyezve, hanem mindnyájan az MDI ügyfél ablakának gyermekei, amely

pedig a keretablak gyermeke. Így a gyermekablakokat a keret „unokáinak” is tekinthetjük;

Ezzel a megoldással több gyermekablakot is megnyithatunk egyszerre, kis méretűre állíthatjuk, bezárhatjuk, rendezhetjük őket a főablakon belül.

A következő ábra és kódrészlet ennek működését mutatja be Mivel programomban a főablak csak a keretet adja az egyes munkafolyamat számára, így minden – az MPS fájl feldolgozásához kapcsolódó – feladat a gyermekablakokban kerül megoldásra. Ezért a program indításakor a felhasználó alapvetően csak két tevékenység közül választhat: elkezdi a munkát egy konkrét MPS fájllal, vagy elindítja a „Súgó” részét képező bemutatót, hogy bővebb információhoz jusson a program használatáról.

A folyamatábra:

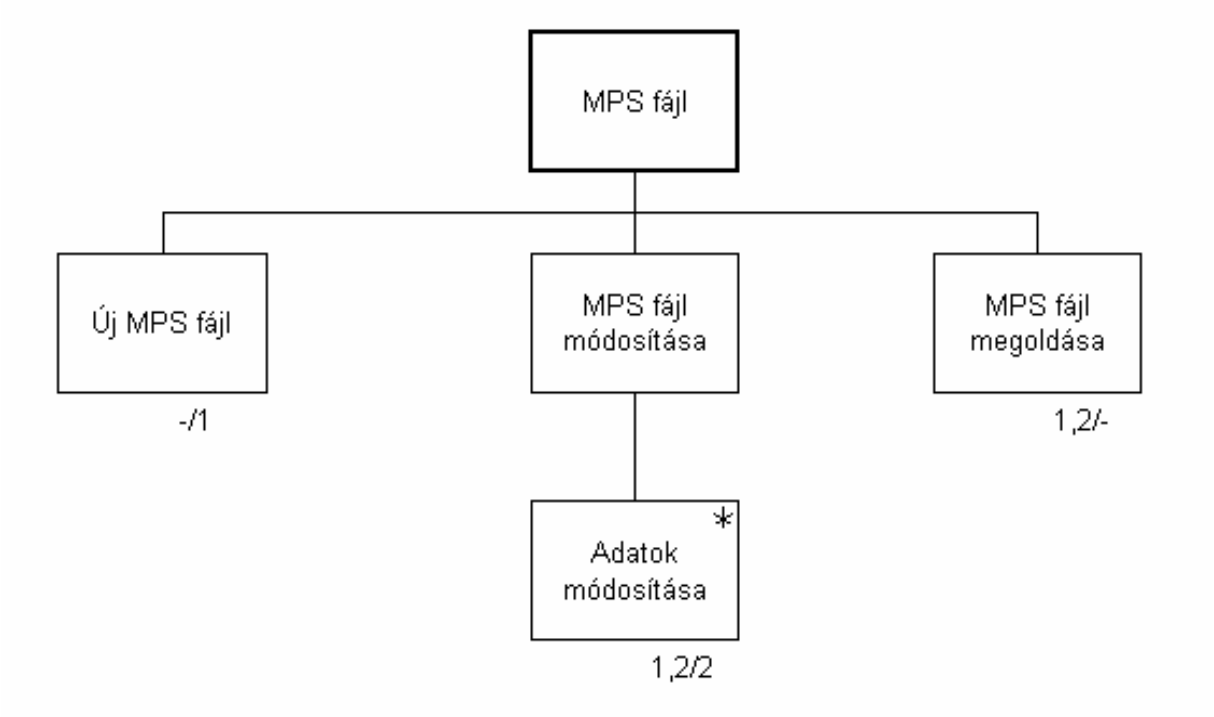


A programkód:

```
var
  MDIChild: TMDIChild;

{$R *.DFM}

implementation

uses main, gombok, lindo, solve, eszkozok;

procedure TMDIChild.FormCreate(Sender: TObject);
begin
  //Kezdő értékek beállítása:
  selRow := 1; selCol := 1;
  NAME := 0; ROWS := 1; COLUMNS := 2; RHS := 3;
  RANGES := 4; BOUNDS := 5; ENDATA := 6;
  myGrid.ColCount := 6;
  myGrid.RowCount := 2;
  myGrid.FixedCols := 0;
  myGrid.FixedRows := 1;
  lMegoldhato := false;

  setCaption;
end;
```

5.2.3. Új MPS fájl létrehozása

Ez a folyamat új MPS fájl elkészítését teszi lehetővé. Mivel elsődleges cél volt, hogy oktatási célú szoftver készüljön, ezért a program felkínálja a felhasználó számára az eredeti IBM szabvány struktúrát, gyakorlatilag azt a mátrix elrendezést, amit a *3.3. fejezetben* tárgyaltam. A felhasználónak „már csak” a megfelelő értékeket kell megadnia a mátrix megfelelő cellájában. A megvalósult képernyőt az *5.2.1. ábra* mutatja.

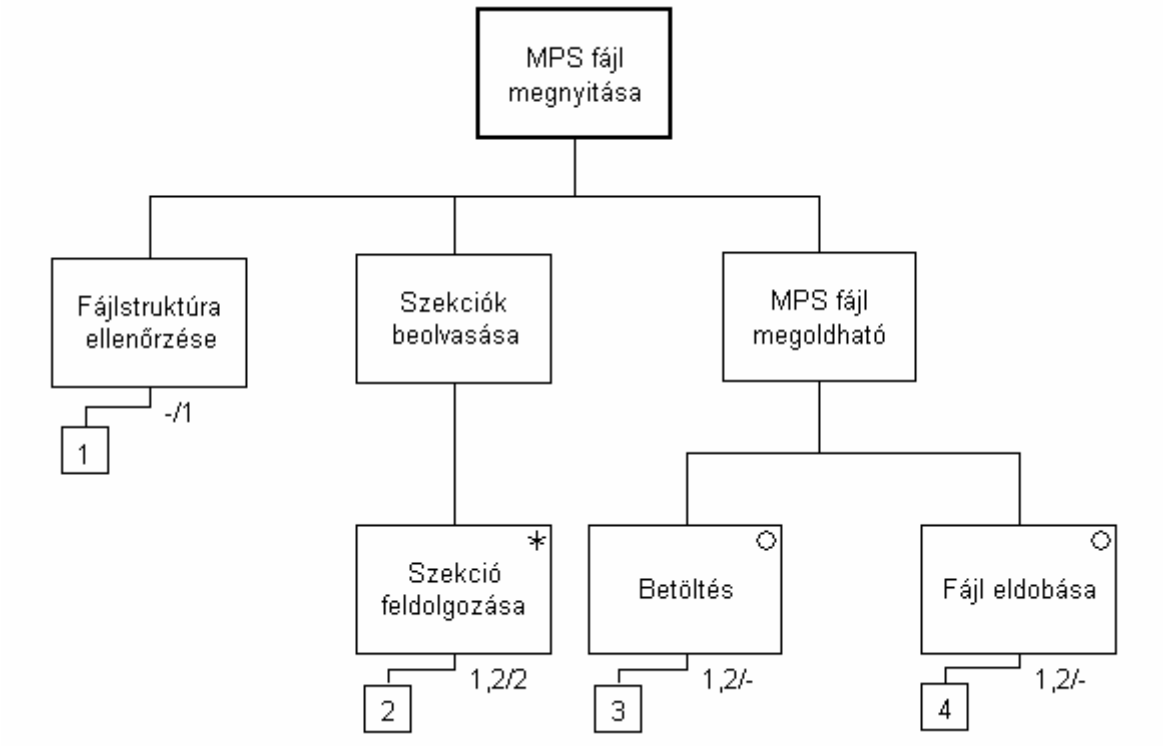


Delphi-ben a mátrix kezelésére több beépített objektum áll rendelkezésünkre (pl. StringGrid, DataGrid). Én a StringGrid-et választottam, mivel ez általános célú adatbevitelt tesz lehetővé. A programkód (a „myGrid” objektum a szerkesztő felület, Delphiben TStringGrid típusú változó, az „aSzekcio” a szabvány szerinti hét darab szekció meglétét nyilvántartó, logikai típusú elemeket tartalmazó tömb):

```
function TMDIChild.makeEmptyGrid: Boolean;
  var i: Integer;
begin
  // Definíció szerinti szekciók:
  myGrid.RowCount := 13;
  myGrid.Cells[0,1] := 'NAME';
  myGrid.Cells[0,2] := 'ROWS';
  myGrid.Cells[0,4] := 'COLUMNS';
  myGrid.Cells[0,6] := 'RHS';
  myGrid.Cells[0,8] := 'RANGES';
  myGrid.Cells[0,10] := 'BOUNDS';
  myGrid.Cells[0,12] := 'ENDATA';

  // Egy álnév a fájlnak:
  myGrid.Cells[2,1] := 'MPS_1';

  // Szekciók inicializálása:
  for i := 0 to 6 do
    aSzekcio[i] := true;

  // Szekciók beszúrása ekkor nem lehetséges:
  for i := 0 to 3 do
    sectionPopUpMenu.Items.Items[i].Enabled := false;

  Result := true;
end;
```

5.2.4. Meglévő MPS fájl beolvasása

Ez a folyamat egy korábban – akár egy egyszerű szövegszerkesztő programmal – készített MPS kiterjesztésű fájl beolvasását valósítja meg. Mivel a formátuma ezeknek a fájloknak kötött, ezért első lépésben ellenőrizni kell, hogy megfelel-e az IBM szabványnak a megnyitandó fájl szerkezete. Ha igen, beolvasásra kerül egy változóba – ebben lesz sorról sorra feldolgozva -, majd a szerkesztési felületet megvalósító StringGrid objektumba (lásd az előző fejezetet). Végül pedig kiíratásra kerül a képernyőre. A folyamat ábrája:



Az egyes operációk – műveletek – jelentése:

1 – Szerepelnek-e a kötelező szekciók a megnyitandó MPS fájlban?

2 – Az egyes szekciók sorainak ellenőrzése, beolvasása.

3 – Betöltés egy munkaállományba

4 – Ha bármilyen ok miatt nem megoldható a feladat, üzenet küldése a felhasználónak.

A programkódok:

1. Először megnézzük, jó formátumú-e a beolvasandó fájl (a kivételkezelés indítása – „try” kulcsszóval - már itt indokolt, hiszen a beolvasás során többször is számátalakításokat kell végezni, ami könnyen memóriabeli számábrázolási hibát eredményezhet):

```
function TMDIChild.MPSFromFile: Boolean;
  var i, szokoz, gridSor: Integer;
      sor, betu: String;
begin
  try
    //A txt-ből beolvasott sorok számának és
    //a megoldhatóság inicializálása:
    i := 0;
    lMegoldhato := false;

    //Keressük a NAME szekciót:
    sor := Trim(MPSText.Lines[0]);
    while ((i <= MPSText.Lines.Count) and
           (not validRow(sor))) do
      begin
        i := i + 1;
        sor := MPSText.Lines[i];
      end;
```

2. A „NAME” kötelező szekciónak keresési eredménye az lesz, hogy az „aSzekcio” tömb – lásd előző fejezet – „NAMES” értéke „igaz” lesz, így a feldolgozás tovább folytatódhat. Ellenkező esetben kilépünk a ciklusból, az adott MPS fájl nem feldolgozható formátumú:

```
//NAME szekció beolvasása:
if Pos('NAME', sor) <> 0 then
  begin
    myGrid.Cells[0, 1] := 'NAME';
    Delete(sor, 1, 5);
    myGrid.Cells[2, 1] := Trim(sor);
    aSzekcio[NAME] := true;
    i := i + 1;
  end;
//A mátrix második sorát kezdjük kitölteni:
myGrid.RowCount := 2;
if ((MPSText.Lines.Count > 0) and (aSzekcio[NAME])) then
  begin
    gridSor := 3;
    // Ciklus a text fájl végéig:
    while ((i <= MPSText.Lines.Count)) do
      begin
        sor := Trim(MPSText.Lines[i]);
        i := i + 1;
```

3. Beolvassuk a soron következő szekciókat (itt most hely hiányában csak a „ROWS” szekciót mutatom be:

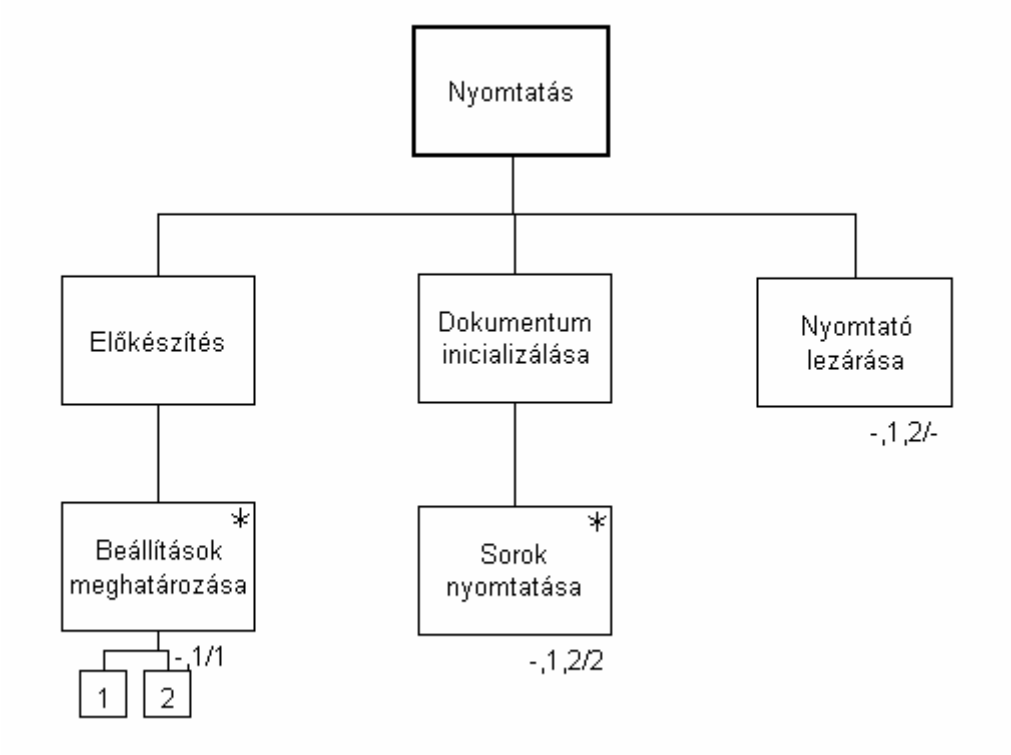
```
// ROWS szekció beolvasása:
if ((Pos('ROWS', sor) <> 0) and (not aSzekcio[ROWS])) then
  begin
    myGrid.RowCount := myGrid.RowCount + 1;
    aSzekcio[ROWS] := true;
    myGrid.Cells[0, 2] := 'ROWS';
    sor := Trim(MPSText.Lines[i]);
    while (((i <= MPSText.Lines.Count) and
            (not validRow(sor))) or
            (sor = '')) do
      begin
        i := i + 1;
        sor := Trim(MPSText.Lines[i]);
      end;
    while ((Pos('COLUMNS', sor) = 0) and
            (i <= MPSText.Lines.Count) and
            (sor <> '')) do
      begin
        betu := Copy(sor, 1, 1);
        Delete(sor, 1, 1);
        if (not joBetu('ROWS', betu)) then
          myMsg(115, MainForm.eleresiUt)
        else
          begin
            myGrid.Cells[0, gridSor] := betu;
            myGrid.Cells[2, gridSor] := Trim(sor);
          end;
        myGrid.RowCount := myGrid.RowCount + 1;
        i := i + 1; gridSor := gridSor + 1;
        sor := Trim(MPSText.Lines[i]);
```

5.2.5. MPS fájl nyomtatása

Előfordulhat, hogy ki akarjuk nyomtatni az elkészült – vagy éppen a folyamatban lévő – MPS fájlunkat. Delphi-ben az ún. „PrintDialog” vizuális komponens áll rendelkezésünkre ennek megoldására. Ez az eszköz az alap Windows nyomtatási dialógus ablak eléréshez teremti meg a kapcsolatot. Addig nem kerül képernyőre ez az ablak, amíg az „Execute” tulajdonságát nem „aktiváljuk”, vagyis amíg nem hívjuk meg futási időben. A komponens tulajdonságaival beállíthatjuk a nyomtatási paramétereket, vagy lehetőséget biztosíthatunk a felhasználónak, hogy ő adja meg – a Windows nyomtatási ablakában – ezeket a paramétereket. (Programomban mindkét lehetőséget kihasználtam: vannak értékek, amiket

nem változtathat meg a felhasználó – pl. nyomtatási tartomány -, és vannak, amiket igen – pl. a nyomtatóválasztás lehetőségének biztosítása).

A folyamatábra:



Az egyes operációk – műveletek – jelentése:

1 – Ideiglenes MPS fájl elkészítése;

2 – Nyomtatási paraméterek meghatározása;

A programkód:

```
// Kinyomtatja a StringGridet grafikus formában:
procedure TMDIChild.printMPS;
  var i, j, sor: Integer;
begin
  // Ideiglenes fájl elkészítése:
  makeFile(MainForm.eleresiUt + '\temp.mps');
  MPSText.Lines.LoadFromFile(MainForm.eleresiUt + '\temp.mps');

  // Nyomtatási opciók meghatározása:
  PrintDialog1.Options := [poPageNums, poSelection];
  PrintDialog1.FromPage := 1;
  PrintDialog1.MinPage := 1;
  if MPSText.Lines.Count > 90 then
    begin
      PrintDialog1.ToPage := Trunc(MPSText.Lines.Count/90) + 1;
      PrintDialog1.MaxPage := Trunc(MPSText.Lines.Count/90) + 1;
    end
  else
    begin
      PrintDialog1.ToPage := 1;
      PrintDialog1.MaxPage := 1;
    end;
  // Nyomtatás
  if PrintDialog1.Execute then
    begin
      with Printer do
        begin
          j := 0;
          // Nyomtatandó dokumentum inicailzálása:
          BeginDoc;
            // Az összes sor nyomtatása:
            for i := 0 to MPSText.Lines.Count do
              begin
                if j = 90 then j := 0; // 90 db sort nyomtatunk 1 oldalra
                sor := 30 + (70 * j);
                // Az aktuális sor nyomtatása:
                Canvas.TextOut(30, sor, MPSText.Lines.Strings[i]);
                j := j + 1;
              end;
          EndDoc;
        end;
    end;
end;
```

5.3. A menürendszer

Programom menürendszerét természetesen a felhasználó által elvégzendő feladatok és azok csoportosítása határozza meg. Főbb felhasználói feladatok:

1. fájlokkal kapcsolatos műveletek;
2. szerkesztéssel kapcsolatos műveletek
3. futtatási környezet beállítási lehetőségei
4. on-line segítség a felhasználónak

A menürendszer összeállítását a Delphi 5-ben egy újabb vizuális komponens, a „MainMenu" segítségével lehet könnyen és gyorsan megvalósítani. Ennek alapján az alábbi menüszerkezetet alakítottam ki:

Fájl menüpont és almenü pontjai:

Új

Üres, IBM szabvány szerinti MPS fájl létrehozása a megfelelő szekciókkal. (Lásd az *5.2.1. ábrát*);

Megnyitás

Meglévő MPS fájl megnyitása

Mentés

Az aktív ablakban lévő MPS fájl mentése, hagyományos text formátumban

Nyomtat

Az aktív ablakban lévő MPS fájl nyomtatása

Megold

Az aktív ablakban lévő MPS feladat megoldása LINDO API segítségével

Bezárás

Az aktív MPS fájl ablak bezárása

Kilépés

Kilépés a programból

Szerkesztés menüpont és almenü pontjai:

Kivágás

Szerkesztés alatt lévő cella tartalmának vágólapra másolása úgy, hogy az eredeti helyéről kivágásra kerül

Másolás

Szerkesztés alatt lévő cella tartalmának vágólapra másolása úgy, hogy az eredeti helyen is megmarad a cellatartalom

Beillesztés

Vágólapon lévő adat beszúrása egy szerkesztés alatt lévő cellába

Ablak menüpont és almenü pontjai:

Egymás fölé

Ha egyszerre több ablakban – több MPS fájllal – dolgozunk, ezt a menüpontot választva egymásra helyezve rendezi a program az ablakokat

Vízszintesen

Ha egyszerre több ablakban – több MPS fájllal – dolgozunk, ezt a menüpontot választva egymás alá/fölé rendezi a program az ablakokat

Beállítások menüpont és almenü pontja:

Futtatási környezet

Ebben a menüpontban lehet beállítani a program működésére vonatkozó paramétereket, úgymint a program nyelvét, az MPS feladatban megjelenítendő értékek tizedes jegy pontosságát, és egy üdvözlő ablak megnyitásának engedélyezését, ami a program indításakor indulhat el. (Lásd az *5.3.1. ábrát*);

Súgó menüpont és almenü pontjai:

Tartalomjegyzék

A súgó indítása

Bemutató futtatása

Néhány hasznos tanács a program használatához

Impresszum

Névjegy a programról

(Az egyes feladatokat el lehet végezni ún. gyorsbillentyűk segítségével, illetve az eszköztár sávon látható ikonok használatával is.)

Az SSADM eszközei közül a „Felhasználó szerep/funkció” mátrixszal ábrázolva a folyamatábrát lásd az 5.*3.2. ábrán*.

5.4. Feladatmegoldás

Kétségtelen, hogy a program elkészítése során a legnagyobb kihívást a LINDO API lehetőségeinek integrálása jelentette. Ahogy azt a *4.3.1. fejezetben* írtam, számos, dll – Dynamic Linked Library - formájában szállított függvény áll rendelkezésre operációkutatási feladatok megoldására ebben az eszközrendszerben. (Lásd még a *4.3.2. fejezetet* a munkamenet megvalósítására.) Gyakorlatilag nem kellett mást tennem, mint a LINDO dokumentációját követni az integrálás megvalósítása érdekében, természetesen a Delphi szintaktikájával. A következő fejezetben részletesebben bemutatom az eljárást.

5.4.1. Megoldás LINDO API-val

Amint az előző részben említettem, az egy-egy MPS fájlban megfogalmazott operációkutatási feladat megoldását megvalósító LINDO API integrálásakor nem kellett mást tennem, mint megfelelő sorrendben meghívnom azokat a külső eljárásokat, függvényeket, amik az API-ban találhatók. Az általános eljárást a *4.3.2. fejezetben* ismertettem egy példán keresztül, itt már a konkrét megvalósítást mutatom meg Delphi 5-ben.

1. Létrehozzuk a LINDO környezetet és az abba ágyazott modellt:

```
// LINDO API integrálása:
procedure TMDIChild.solveMPS(Sender: TObject);
var szLicence, szMPSFile: String;
    nEnv,nModel: Pointer;
    obj: Double;
    adX: Array of Double;
    is_mip, errorCode, status, nvars, ncons, ncont, nonz: Integer;
begin
  // Ha korábban már volt megoldás, töröljük az output fájlt:
  if FileExists('out.txt') then DeleteFile('out.txt');
  // Licensz információk olvasása:
  SetLength(szLicence, 1024);
  LSloadLicenseString(pchar(MainForm.eleresiUt +
                            '\lndapi50.lic'),pchar(szLicence));
  // LINDO környezet létrehozása:
  nEnv := LScreateEnv (errorcode, pchar(szLicence));
  ErrorCheck(nEnv, errorCode);
  // LINDO modell létrehozása:
  nModel :=  LScreateModel (nEnv,errorcode) ;
  ErrorCheck(nEnv, errorCode);
```

2. Beolvassuk az MPS fájlt:

```
// Munkamenet MPS fájl beolvasása:
szMPSFile := MainForm.eleresiUt + '\temp.mps';
errorCode := LSreadMPSFile(nModel, pchar(szMPSFile),
                           LS_UNFORMATTED_MPS);
ErrorCheck(nEnv, errorCode);
// Ha MPS formátum nem megy, lehet még próbálkozni
// LINDO és MPI formátummal:
if (errorcode <> LSERR_NO_ERROR) then
  begin
    errorCode := LSreadLINDOFile(nModel, pchar(szMPSFile));
    if (errorcode <> LSERR_NO_ERROR) then
      begin
        errorCode := LSreadMPIFile(nModel, pchar(szMPSFile));
        ErrorCheck(nEnv, errorCode);
      end;
    end;
```

3. Ha a beolvasás sikeres volt, kiolvassuk az adott feladat értékeit:

```
// Ha a beolvasás sikers volt:
if (errorcode = LSERR_NO_ERROR) then
  begin
    // Információk a beolvasott modellről:
    errorCode := LSgetInfo(nModel, LS_IINFO_NUM_VARS, nvars);
    ErrorCheck(nEnv, errorCode);
    errorCode := LSgetInfo(nModel, LS_IINFO_NUM_CONS, ncons);
    ErrorCheck(nEnv, errorCode);
    errorCode := LSgetInfo(nModel, LS_IINFO_NUM_NONZ, nonz);
    ErrorCheck(nEnv, errorCode);
    errorCode := LSgetInfo(nModel, LS_IINFO_NUM_CONT, ncont);
    ErrorCheck(nEnv, errorCode);
    // Információk megjelnítése:
    labcon.Caption := IntToStr(ncons);
    labvar.Caption := IntToStr(nvars);
    labint.Caption := IntToStr(nvars-ncont);
    labnonz.Caption := IntToStr(nonz);
```

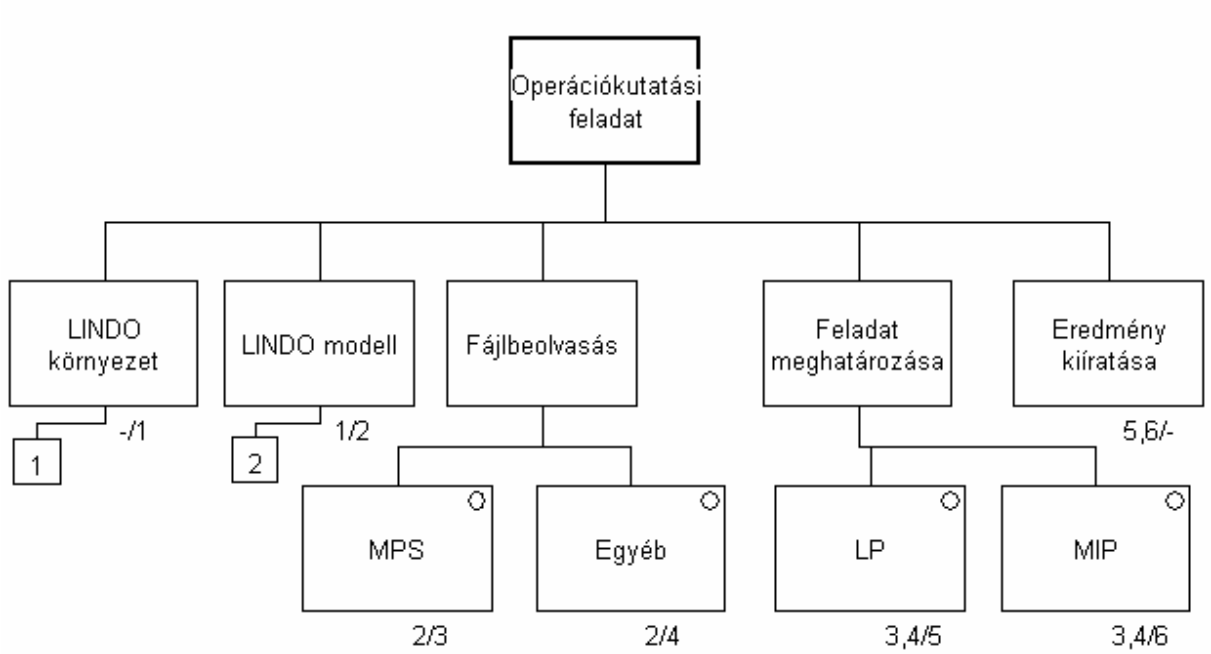
4. Megoldjuk a feladatot:

```
// Ha a változók és konstansok száma egyenlő,
// folytonos feladatot kell megoldani:
if (ncont = nvars) then is_mip := 0
else is_mip := 1;
if (is_mip = 0) then
  begin
    // Ha a feladat folytonos:
    errorCode := LSoptimize(nModel, LS_METHOD_FREE, status);
    errorCode := LSgetInfo(nModel, LS_DINFO_POBJ, obj);
    // Primál tömb hosszának beállítása:
    setLength(adX,nvars);
    errorCode := LSgetPrimalSolution(nModel, adX[0]);
  end
else
  begin
    // Ha a feladat egészértékű:
    errorCode := LSSolveMIP(nModel, status) ;
    errorCode := LSgetInfo(nModel, LS_DINFO_MIP_OBJ, obj);
    errorCode := LSgetMIPPrimalSolution(nModel, adX[0]);
  end;
```

5. Ha volt megoldás, elkészül az output fájl (lásd 1.2. ábra*), majd fel kell szabadítani a lefoglalt memóriát, és az eredményt ki kell íratni a képernyőre (lásd az* 5.4.1. ábrát*):*

```
    // Output fájl létrehozása:
    errorCode := LSwriteSolution(nModel, 'out.txt');
  end;

// LINDO környezet felszabadítása:
LSdeleteEnv(nEnv);
// Ha volt megoldás, betöltés egy újabb űrlapra:
if FileExists('out.txt') then
  begin
    SolveForm := TSolveForm.Create(Self);
    SolveForm.Caption := 'out.txt';
    SolveForm.SolveReport.Lines.LoadFromFile('out.txt');
    SolveForm.ShowModal;
    SolveForm.Release;
  end;
```

A feladatmegoldás SSADM ELH diagrammja:



Az egyes operációk – műveletek – jelentése:

1 – LINDO környezet létrehozása;

2 – LINDO modell létrehozása a LINDO környezetben;

5.5. Technikai információk

Szakdolgozatom programját a Borland cég Delphi 5 Enterprise verziójú fejlesztőeszközével készítettem, amelynek futtatásához a Windows operációs rendszer 32 bites verziója szükséges (Windows NT, Windows ME, Windows 2000, Windows XP). Ebből következik, hogy olyan hardver kell a programom futtatásához, amin az említett operációs rendszer már képes futni. (Kiskereskedelmi forgalomban gyakorlatilag már ettől kisebb „tudású” PC-t, notebook-ot nem lehet vásárolni.) Speciális hardver eszközre nincs szükség. Az integrált LINDO API függvényeket tartalmazó dll fájlokat a telepítőkészlethez csatoltam (lásd *5.5.1. fejezet*) – ezek ingyenesen hozzáférhető termékek a LINDO internetes oldaláról -, ezen kívül speciális szoftverre nincs szükség a program futtatásához.

5.5.1. Telepítés

Napjainkban elengedhetetlen, hogy egy szoftver, mint informatikai termék felhasználóbarát legyen a telepítéstől kezdve az esetleges számítógépről történő eltávolításáig. Egy programfejlesztőknek szánt eszköztől elvárható, hogy rendelkezzen beépített telepítőkészlet készítő, varázsló modullal. A Delphi 5 Enterprise verziója rendelkezik ilyennel, az InstallShield Express nevű programmal. Én mégis egy nyílt forráskódú terméket, az Inno Setup programot választottam a telepítőprogram elkészítéséhez (egyébként ezt is Delphi-ben írta a készítője, lásd az *Irodalomjegyzéket*). Ennek oka, hogy ezzel az eszközzel lehet magyar nyelvű telepítő programcsomagot is előállítani, míg a Delphi modulja csak az angol nyelvet támogatja. (A telepítő csomag elkészítéséről nem kívánok szólni, mert az nem része a szakdolgozatomnak)

A szakdolgozatom fizikai mellékletét képezi egy telepítő CD, amin a következő könyvtárstruktúra található:

Könyvtár	Alkönyvtár	Fájl	Leírás
Doc		Lindoapi.pdf	A LINDO API teljes angol nyelvű dokumentációja
		Dolgozat.pdf	Ez a szakdolgozat
Hdi		Conopt3.dll	LINDO API fájl
		Dformd.dll	LINDO API fájl
		Hdi.dat	A HDI 1.0 program inicializációs fájlja
		Hdi.exe	A program
		Libguide40.dll	LINDO API fájl
		Lindo5_0.dll	LINDO API fájl
		Lndapi50.lic	LINDO API licensz fájl
		Mosek5_0.dll	LINDO API fájl
		Utils.dll	A HDI 1.0 program üzeneteit kezelő modulja
	Bitmaps	No.bmp	Figyelmeztető hibaüzenetek ikonja
		Question.bmp	Kérdés üzenetek ikonja
		Stop.bmp	Javítandó hibaüzenetek ikonja
		Yes.bmp	Folyamat lezárását nyugtázó üzenetek ikonja
	Help	Hdi.cnt	Súgó tartalomjegyzéke

		Hdi.hlp	A súgó fájl
	Lang	Hdi_msg_en.dat	Képernyőre kerülő angol nyelvű szövegek
		Hdi_msg_hu.dat	Képernyőre kerülő magyar nyelvű szövegek
	Sample	Sajat.mps	A programmal előállított teszt MPS fájl
		Testlp.mps	LP feladat MPS fájlja
		Testmip.mps	MIP feladat MPS fájlja
Install		Setup.exe	A telepítő program
Prg		Prg.rar	A tömörített forrásprogram, jelszóval védve

A telepítéshez rendszergazda jogok szükségesek a célszámítógépen. Erre azért van szükség, mert a telepítő program a Windows Registry fájljába bejegyzi a programot a későbbi támogatás eléréséhez, ill. az esetleges eltávolításhoz. Ha azonban nincs rendszergazda jogunk a célszámítógépen, a CD-n lévő *\hdi* könyvtár teljes tartalmát felmásolva az adott gép fizikai háttértárolójára, probléma nélkül tudjuk használni a programot. (Ekkor azonban kézzel kell gondoskodni az esetleges eltávolításról.)

A telepítés menete (kb. 30 Mb szabad helyre van szükség a célszámítógépen, a telepítés ideje kevesebb, mint 1 perc):

1. Célkönyvtár megadása (alapértelmezett a C:\Program files\hdi könyvtár);
2. A „Start” menü mappájának megadása (alapértelmezett a HDI);
3. „Asztal”-ra kerülő parancsikon létrehozásának eldöntése (alapértelmezésben nem jön létre parancsikon);
4. A HDI program indításának eldöntése (alapértelmezésben elindul a HDI program;

6. Összefoglalás

Szakdolgozatom elkészítésével nem egy újabb programot akartam készíteni egy-egy operációkutatási feladat megoldására, hanem az adott feladat alapját jelentő fájlformátum kezelését akartam könnyebbé tenni.

Meglepő módon – ahogy azt a *Bevezetőben* is jeleztem – nem találtam MPS fájlkezelő – készítő, módosító – programot. Abban a formában legalábbis nem, ahogy azt az IBM szabvány tárgyalja. Az egyik legnagyobb operációkutatási szoftvert gyártó cég – a LINDO, aminek én is felhasználtam a termékét – a Windows/Linux platform alá írt programja természetesen kezeli az MPS fájlformátumot, de nem az IBM szintaktikával.

Ebből kiindulva programom hiánypótló szoftver, természetesen a továbbfejlesztés lehetőségeinek kihasználásával. Egy továbbfejlesztési lehetőség például a Linux platformra történő fejlesztés, ami a programmal szemben támasztott hordozhatósági követelménynek tenne eleget.

A program elkészítése során folyamatosan teszteltem a megoldható operációkutatási feladatok eredményeit a programom által szolgáltatott és a LINDO által szolgáltatott eredmények összehasonlításával. A tesztelésre tehát kétféle módszert választottam:

1. LINDO által előállított MPS formátumú fájlt oldottam meg a programommal (lásd a csatolt CD-n lévő program .”.\Sample\testlp.mps” fájlt);
2. A programom által előállított MPS fájlt oldattam meg a Lingo 10.0 verziójával (lásd a csatolt CD-n lévő program .”.\Sample\sajat.mps” fájlt);

A tesztelések eredménye mindkét esetben ugyanaz lett (Az eredményekről lásd még az *Ábrajegyzék* 1.1., 1.2. és 1.3. ábrákat.)

Szakdolgozatomban igyekeztem bemutatni a fontosabb alapjait egy operációkutatási feladatot leíró fájl formátum kezeléséhez szükséges programnak (a 2. és a 3. fejezetben), a megoldást megvalósító, külső gyártó által elkészített algoritmusok integrálásának (a 4. fejezetben), végül pedig a program elkészítésének (5. fejezet).

Irodalomjegyzék

LINDO API dokumentáció:
LINDO API 5.0 User Manual
LINDO System Inc., 2007

Lukács Ottó: Operációkutatás
Tankönyvkiadó, Budapest, 1982

J. Stanley Warford: Computer Science, Volume 1
D.C. Heath and Company, Lexington, Massachusetts/Toronto, 1991.

Marco Cantù: Delphi 5 Mesteri szinten, I-II kötet
Kiskapu Kft., Budapest, 2000.

LP feladatok leírása:
www.stud.u-szeged.hu/Jenei.Peter.2/jegyzetek/opkut.doc

Lineáris Programozás:
http://www.bke.hu/puskas/elibd/szimplex.pdf

Komáromi Éva, ”Operációkutatás”, 2005
http://web.uni-corvinus.hu/~opkut/elibd/linearis_programozas.pdf

MPS input format:
http://www-fp.mcs.anl.gov/otc/Guide/OptWeb/continuous/constrained/linearprog/mps.html

MPS format:
http://en.wikipedia.org/wiki/MPS_%28format%29

Delphi – Marco Cantù weboldala:
http://www.marcocantu.com/

Inno Setup – Telepítő fájlkészítő eszköz:
http://www.palkornel.hu/innosetup/

Ábrajegyzék

1.1. ábra: a HDI 1.0 programhoz mellékelt „testlp.mps” fájl:

```
NAME   AFIRO
ROWS
E   R09
E   R10
L   X05
L   X21
E   R12
E   R13
L   X17
L   X18
L   X19
L   X20
E   R19
E   R20
L   X27
L   X44
E   R22
E   R23
L   X40
L   X41
L   X42
L   X43
L   X45
L   X46
L   X47
L   X48
L   X49
L   X50
L   X51
N   COST
COLUMNS
 X01 X48 0.3010 R09 -1.0000
 X01 R10 -1.0600 X05 1.0000
 X02 X21 -1.0000 R09 1.0000
 X02 COST -0.4000
 X03 X46 -1.0000 R09 1.0000
 X04 X50 1.0000 R10 1.0000
 X06 X49 0.3010 R12 -1.0000
 X06 R13 -1.0600 X17 1.0000
 X07 X49 0.3130 R12 -1.0000
 X07 R13 -1.0600 X18 1.0000
 X08 X49 0.3130 R12 -1.0000
 X08 R13 -0.9600 X19 1.0000
 X09 X49 0.3260 R12 -1.0000
 X09 R13 -0.9600 X20 1.0000
 X10 X45 2.3640 X17 -1.0000
 X11 X45 2.3860 X18 -1.0000
 X12 X45 2.4080 X19 -1.0000
 X13 X45 2.4290 X20 -1.0000
 X14 X21 1.4000 R12 1.0000
 X14 COST -0.3200
 X15 X47 -1.0000 R12 1.0000
 X16 X51 1.0000 R13 1.0000
 X22 X46 0.1090 R19 -1.0000
 X22 R20 -0.4300 X27 1.0000
 X23 X44 -1.0000 R19 1.0000
 X23 COST -0.6000
 X24 X48 -1.0000 R19 1.0000
 X25 X45 -1.0000 R19 1.0000
 X26 X50 1.0000 R20 1.0000
 X28 X47 0.1090 R22 -0.4300
 X28 R23 1.0000 X40 1.0000
 X29 X47 0.1080 R22 -0.4300
 X29 R23 1.0000 X41 1.0000
 X30 X47 0.1080 R22 -0.3900
 X30 R23 1.0000 X42 1.0000
 X31 X47 0.1070 R22 -0.3700
 X31 R23 1.0000 X43 1.0000
 X32 X45 2.1910 X40 -1.0000
 X33 X45 2.2190 X41 -1.0000
 X34 X45 2.2490 X42 -1.0000
 X35 X45 2.2790 X43 -1.0000
 X36 X44 1.4000 R23 -1.0000
 X36 COST -0.4800
 X37 X49 -1.0000 R23 1.0000
 X38 X51 1.0000 R22 1.0000
 X39 R23 1.0000 COST 10.0000
RHS
 B X50 310.0000 X51 300.0000
 B X05 80.0000 X17 80.0000
 B X27 500.0000 R23 44.0000
 B X40 500.0000
ENDATA
```

1.2. ábra: az 1.1. ábrán szereplő feladat – AFIRO – eredménye a HDI 1.0 programmal, LINDO API függvényekkel:

```
---------------------------------------------------------------------
            SOLUTION REPORT

LINDO API Version 5.0.1.193 built on Oct 26 2007 21:23:19
Barrier Solver Version 5.0.0.062, Nonlinear Solver Version 3.14R

Copyright (c) 2007 by LINDO Systems, Inc. Licensed material,
all rights reserved. Copying except as authorized in license
agreement is prohibited.
---------------------------------------------------------------------

PROBLEM NAME     AFIRO

LP GLOBAL OPTIMUM FOUND

ITERATIONS BY SIMPLEX METHOD =         3
ITERATIONS BY BARRIER METHOD =         0
ITERATIONS BY NLP METHOD     =         0
TIME ELAPSED (s)             =         0

OBJECTIVE FUNCTION VALUE

1)          -464.753142857

VARIABLES              VALUE          REDUCED COST          BASIS STATUS

X01              80.000000000         0.000000000        LS_BASTYPE_BAS
X02              25.500000000         0.000000000        LS_BASTYPE_BAS
X03              54.500000000         0.000000000        LS_BASTYPE_BAS
X04              84.800000000         0.000000000        LS_BASTYPE_BAS
X06              18.214285714         0.000000000        LS_BASTYPE_BAS
X07               0.000000000         2.249657143        LS_BASTYPE_ATLO
X08               0.000000000         2.270400000        LS_BASTYPE_ATLO
X09               0.000000000         2.290200000        LS_BASTYPE_ATLO
X10               0.000000000         2.228914286        LS_BASTYPE_ATLO
X11               0.000000000         0.000000000        LS_BASTYPE_BAS
X12               0.000000000         0.000000000        LS_BASTYPE_BAS
X13               0.000000000         0.000000000        LS_BASTYPE_BAS
X14              18.214285714         0.000000000        LS_BASTYPE_BAS
X15               0.000000000         0.000000000        LS_BASTYPE_BAS
X16              19.307142857         0.000000000        LS_BASTYPE_BAS
X22             500.000000000         0.000000000        LS_BASTYPE_BAS
X23             475.920000000         0.000000000        LS_BASTYPE_BAS
X24              24.080000000         0.000000000        LS_BASTYPE_BAS
X25               0.000000000         0.000000000        LS_BASTYPE_BAS
X26             215.000000000         0.000000000        LS_BASTYPE_BAS
X28               0.000000000         0.000000000        LS_BASTYPE_ATLO
X29               0.000000000         2.092200000        LS_BASTYPE_ATLO
X30               0.000000000         2.120485714        LS_BASTYPE_ATLO
X31               0.000000000         2.148771429        LS_BASTYPE_ATLO
X32               0.000000000         2.065800000        LS_BASTYPE_ATLO
X33               0.000000000         0.000000000        LS_BASTYPE_BAS
X34               0.000000000         0.000000000        LS_BASTYPE_BAS
X35               0.000000000         0.000000000        LS_BASTYPE_BAS
X36             339.942857143        -0.000000000        LS_BASTYPE_BAS
X37             383.942857143         0.000000000        LS_BASTYPE_BAS
X38               0.000000000         0.000000000        LS_BASTYPE_BAS
X39               0.000000000        10.000000000        LS_BASTYPE_ATLO
```

1.3. ábra: az 1.1. ábrán szereplő feladat – AFIRO - Lingo 10.0 programmal előállított eredménye:

Global optimal solution found.
Objective value: -464.7531

Total solver iterations: 3

Model Title: AFIRO

Variable	Value	Reduced Cost
X02	25.50000	0.000000
X14	18.21429	0.000000
X23	475.9200	0.000000
X36	339.9429	0.000000
X39	0.000000	10.00000
X01	80.00000	0.000000
X03	54.50000	0.000000
X04	84.80000	0.000000
X06	18.21429	0.000000
X07	0.000000	2.249657
X08	0.000000	2.270400
X09	0.000000	2.290200
X15	0.000000	0.000000
X16	19.30714	0.000000
X10	0.000000	2.228914
X11	0.000000	0.000000
X12	0.000000	0.000000
X13	0.000000	0.000000
X22	500.0000	0.000000
X24	24.08000	0.000000
X25	0.000000	0.000000
X26	215.0000	0.000000
X28	0.000000	0.000000
X29	0.000000	2.092200
X30	0.000000	2.120486
X31	0.000000	2.148771
X38	0.000000	0.000000
X37	383.9429	0.000000
X32	0.000000	2.065800
X33	0.000000	0.000000
X34	0.000000	0.000000
X35	0.000000	0.000000

3.1 ábra: MPS lyukkártya formátum

Tartalom		Jelző	Üres	Oszlopnév	Üres	Sornév1	Üres	*Num.érték1*	Üres	Sornév2	Üres	*Num.érték2*	Üres
Pozíció	1	2 - 3	4	5 - 12	13 - 14	15 - 22	23 - 24	25 - 36	37 - 39	40 - 47	48 - 49	50 - 61	62 - 80
1. sor		**NAME**				**Feladat1**							
2. sor		**ROWS**											
3. sor		**L*****		**R1**									
4. sor		**L*****		**R2**									
5. sor		**COLUMNS**											
6. sor				**C1**		**R1**		**12**		**R2**		**14**	
7. sor				**C2**		**R1**		**3**		**R2**		**5**	
8. sor		**RHS**											
9. sor				**R_vektor1**		**R1**		**150**		**R2**		**200**	
10. sor		**RANGES***											
11. sor				**R_vektor2**		**R1**		**R****					
12. sor		**BOUNDS***											
13. sor													
14. sor		**ENDATA**											
Hossz	1	2	1	8	2	8	2	12	3	8	2	12	19

*Nem kötelező!

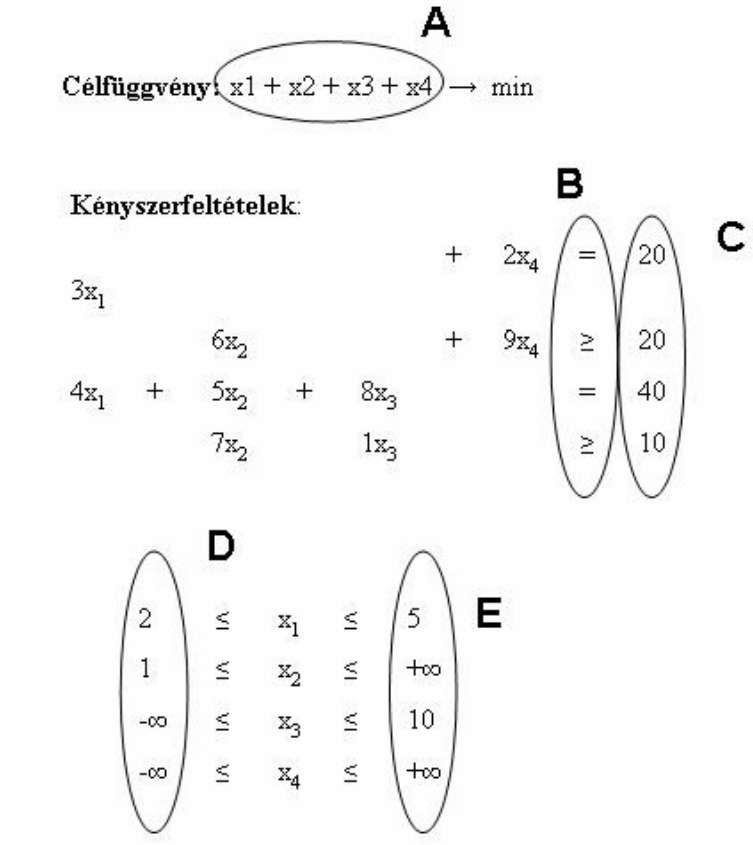
**R értéke: $ha\ L_i \le \sum_{j=1}^{n} a_{ij} x_j \le U_i\ ,\ \ akkor\ R = U_i - L_i$

***Kulcsszavak

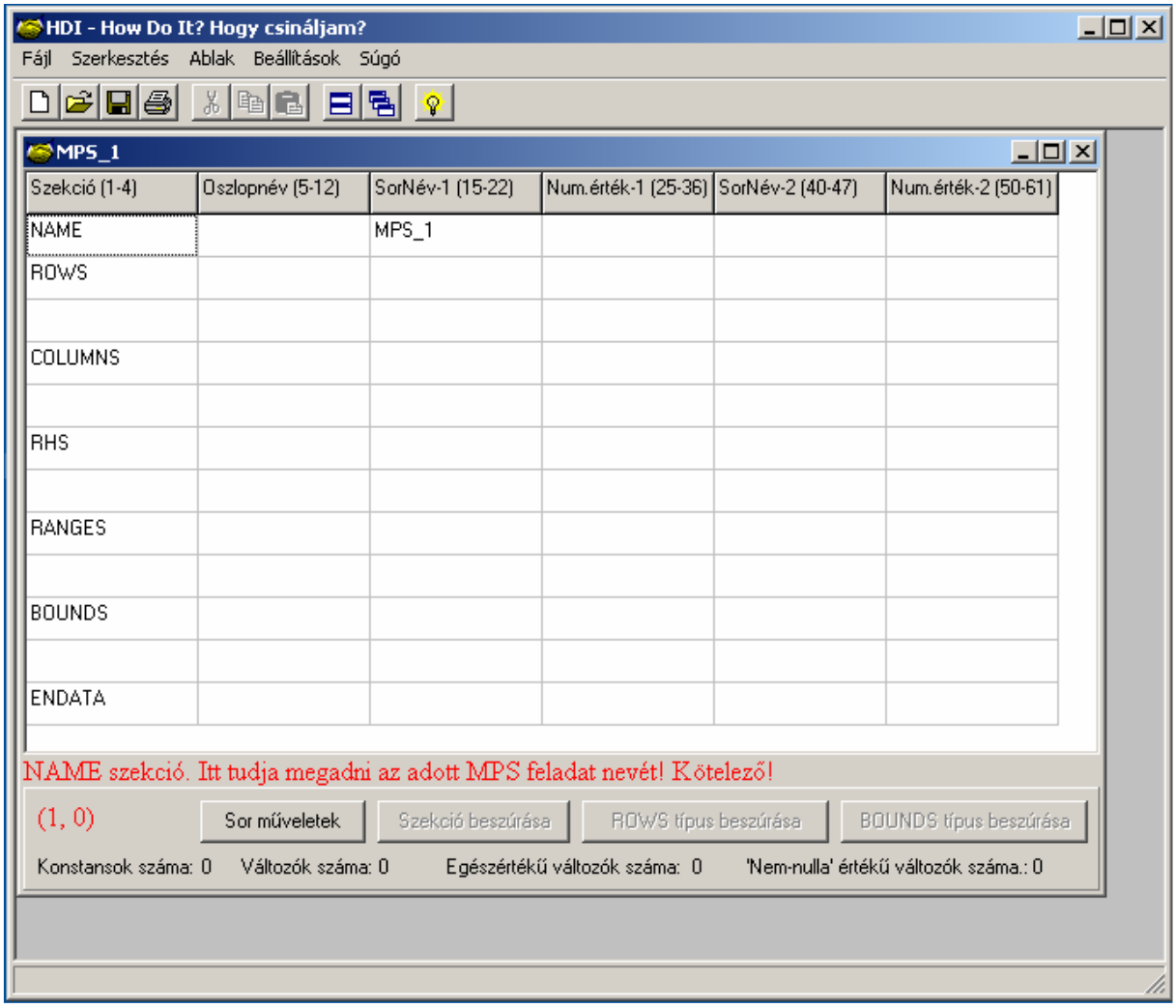
3.3.1. ábra: Példa MPS formátumra

```
NAME    OPKUT
ROWS
L    F1
L    F2
L    F3
N    HASZON
COLUMNS
  x  F1  3.0000  F2  1.0000
  x  F3  1.0000
  x  HASZON  60.0000
  y  F1  3.0000  F2  4.0000
  y  HASZON  30.0000
RHS
  RHS_1  F1  420.0000
  RHS_1  F2  440.0000
RANGES
BOUNDS
UP  BND_1  x  80.0000
ENDATA
```

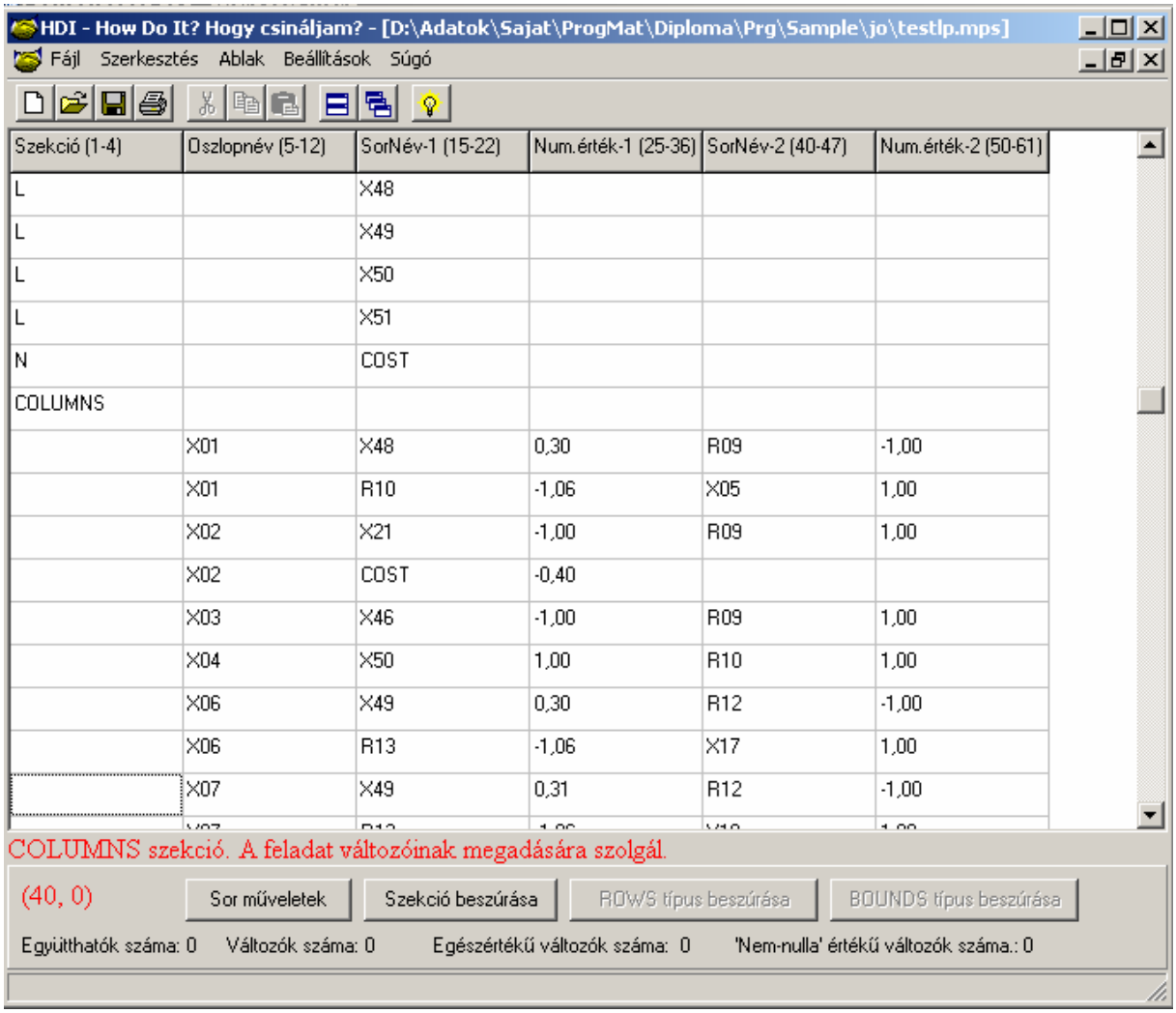
4.1. ábra: egy LP feladat felosztása tömbökre



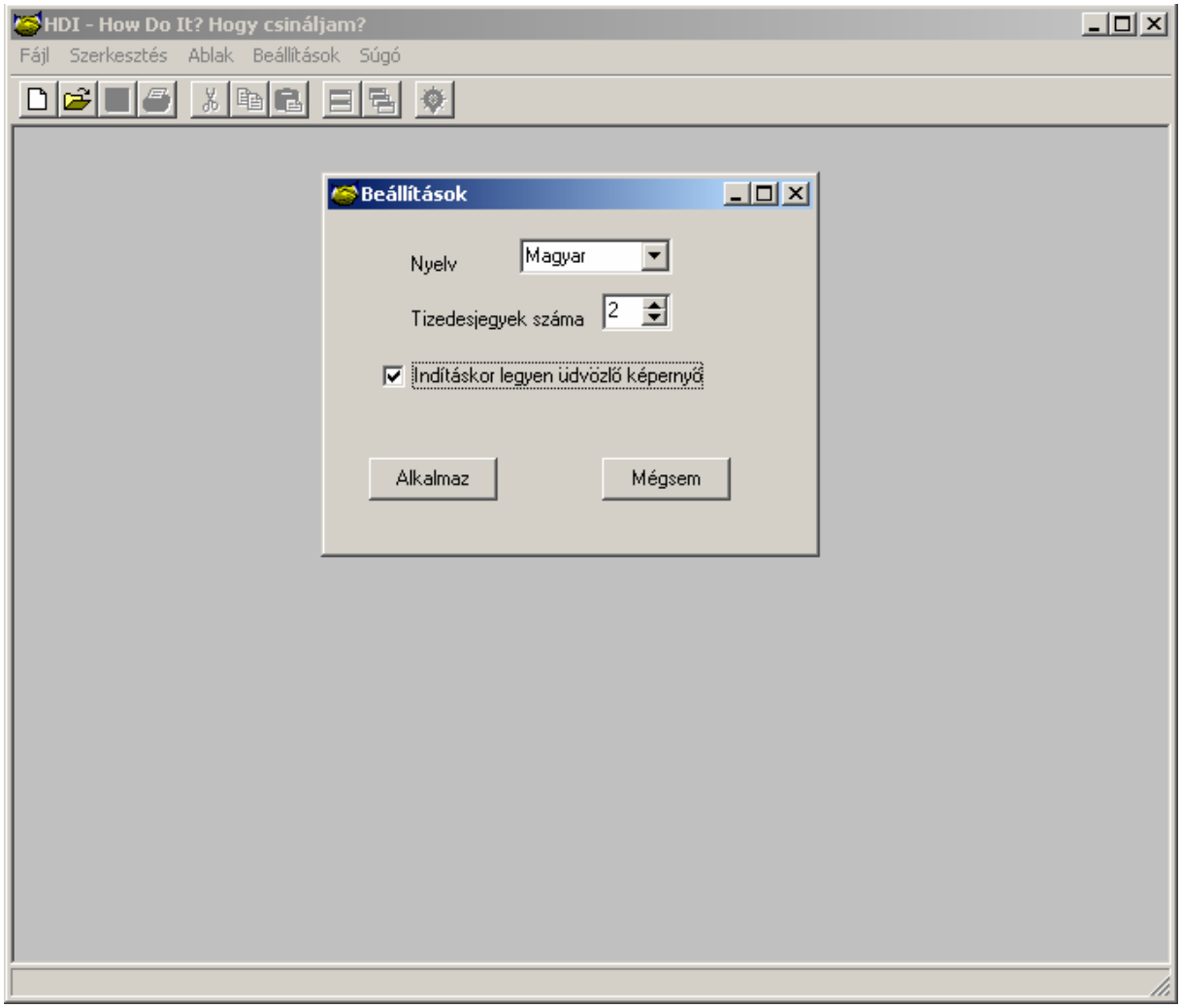
5.2.1. ábra: Új MPS fájl készítése:



5.2.2. ábra: Meglévő MPS fájl szerkesztése

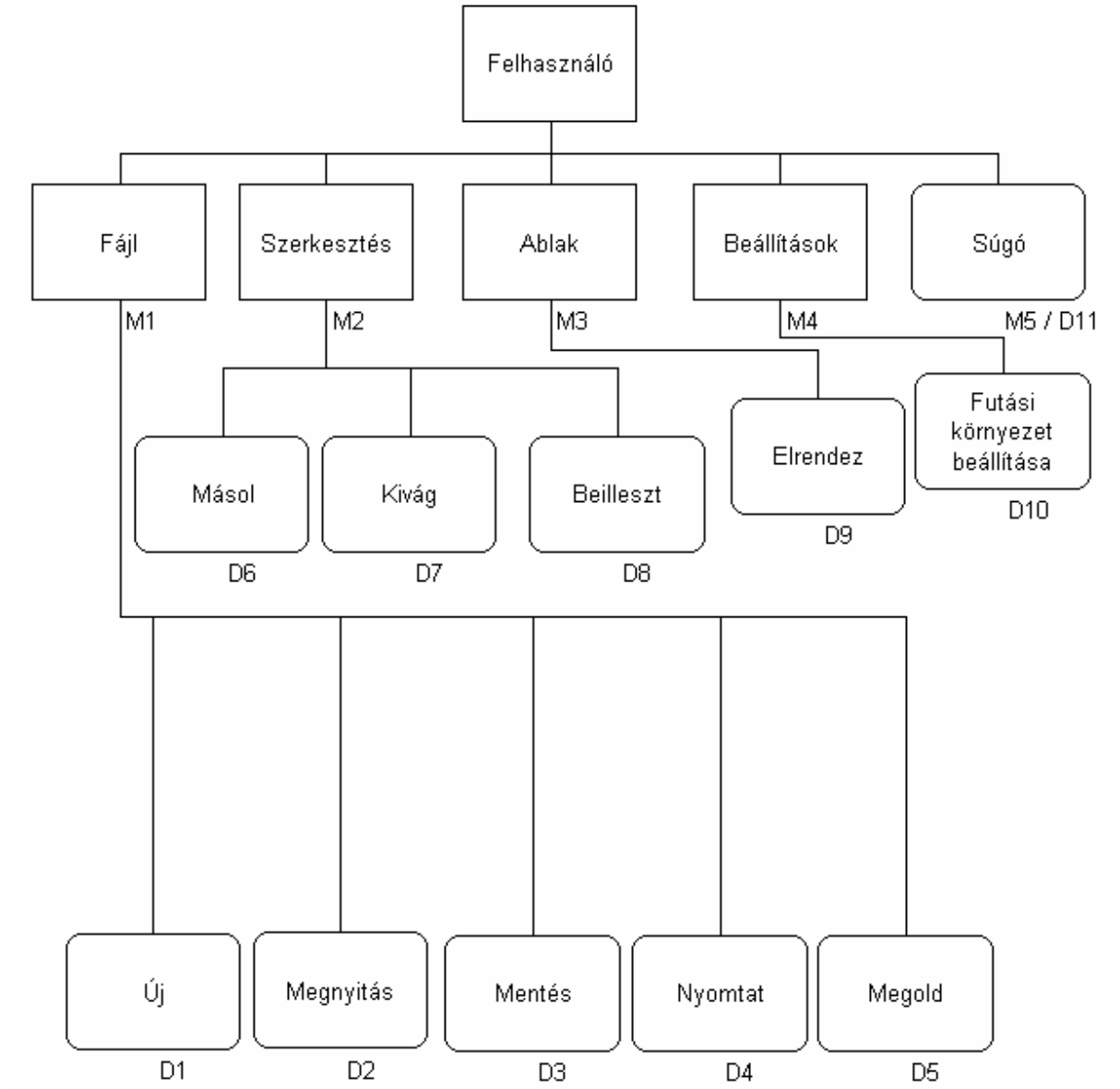


5.3.1. ábra: Környezeti beállítások módosítása



5.3.2. ábra: A HDI program menürendszere SSADM eszközökkel

	Új (D1)	Megnyitás (D2)	Mentés (D3)	Nyomtat (D4)	Megold (D5)	Másol (D6)	Kivág (D7)	Beilleszt (D8)	Elrendez (D9)	Futási környezet beállítása (D10)	Súgó (D11)
Felhasználó	X	X	X	X	X	X	X	X	X	X	X



5.4.1. ábra: A HDI 1.0 programmal előállított operációkutatási feladat LINDO API-val történő megoldásának output képernyője