SZOFTVER

HANÁK GÁBOR

Nagyméretű modell automatizált input-rendszere

A számítógép alkalmazása során igen gyakran merül fel az a jelentős nehézségeket, sőt a nem számítástechnikai szakemberek részéről bizonyos ellenérzéseket és ellenkezéseket felvető probléma, hogy a ragyogóan működő kész programok, programrendszerek vagy programcsomagok életközeli működtetéséhez többnyire rengeteg és nagyon sokféle input adatot kell megadni. Ez a nehézség különösen akkor érezteti hatását, amikor több ezer, tízezer input adatra van szükség. Egy programterméket, illetve annak egy konkrét megvalósítását tulajdonképpen nem is tekinthetünk teljes értékűnek, ha nem tartalmaz — valamilyen mélységig és rugalmassági fokig — automatikus input-output rendszert. A gyakorlati munka sikere szempontjából ugyanis a számítógéptől távol álló szakemberek joggal várják el: nem elég a problémát megoldani, a megoldást be kell ágyazni a környezetbe.

Az egyik legnagyobb nehézség az, hogy az adatszolgáltatóktól elvárható adatrendszer és a programcsomag működtetéséhez szükséges adatrendszer között gyakran óriási szakadék tátong. Ezt ugyan „kézi" módszerekkel is betömhetjük, de csak igen nagy mennyiségű munka árán. Ha azonban az adatrendszer egyes változatainak többszöri átfuttatására van szükség, akkor ez a módszer gyakorlatilag használhatatlan. Szükség van tehát az input-folyamat automatizált, gépi megoldására.

Az elkövetkezőkben kísérletet teszünk a fent jelzett problémával kapcsolatban bizonyos szabályszerűségek végiggondolására és megfogalmazására, továbbá a szükséges mértékben leegyszerűsítve ismertetjük azt a sémát, amelyet egy konkrét esetben dolgoztunk ki. Írásunkban a teljes rendszernek csak az input oldalával foglalkozunk.

1.1 Automatizált input-rendszer

A továbbiakban a következő egyszerűsítő feltevésekkel élünk:

— adott egy *programcsomag*, ami képes arra, hogy a meghatározott formában érkező, ún. *előkészített adatrendszert*, amely a vizsgálandó *modell* egy speciális *menetrendje*, feldolgozza és az eredményeket valamilyen (számunkra most érdektelen) formában közölje;

— az előkészített adatrendszer tartalmaz bizonyos *alapadatok*at, amelyek a modell valamilyen szintű vázát alkotják, s amelyek minden egyes menetrend esetében változatlanok maradnak;

— létezik egy *elsődleges adatrendszer*, amelyből az alapadatokkal együtt, valamilyen szintű feldolgozással, előállítható az *előkészített adatrendszer*, ami minden egyes menetrend esetében más és más lehet.

Az elsődleges és az előkészített adatrendszer megkülönböztetésére azért van szükség, mert a programcsomag a modell feldolgozásához általában más adatokat vár inputként, mint amiket az adatszolgáltatóktól célszerű megkérni. (Például ha a modell által figyelembe vett egyik anyag két másik anyag keverékéből áll, s ezek mindegyike adott összetételű egy menetrend esetében, akkor a keverék összetételét célszerű egy programmal előállítani; vagy ha a modellben több helyen is előfordul egy termék, aminek az árára is szükség van, akkor ezt az egy információt fölösleges minden esetben megkérni: érdemes az egyetlen adatelem sokszorozására programot írni.) A kétféle adatrendszert *algoritmusrendszer* kapcsolja össze, ha az előkészített adatrendszer minden egyes eleméhez van egy olyan program, ami azt az alapadatokból (illetve azok egy részéből) és az elsődleges adatrendszerből (illetve ennek egy részéből) előállítja. Az algoritmusok megadása természetesen részben az adatszolgáltató, részben pedig a programozó feladata. (Az előző példákban: az anyagok összetételét, és azt, hogy ebből a keverékét hogyan kell számolni, az adatszolgáltató köteles megmondani. Az adatszolgáltatónak itt nem programot kell írnia, csak a megfelelő útmutatásokat kell megadnia!); az adatelem sokszorozásáért viszont a programozó a felelős.)

Nyilvánvaló, hogy egy adott modell esetében az alapadatok a többi adattól egyértelműen elhatárolhatók. Az elsődleges adatrendszer kijelölése azonban már korántsem ilyen egyszerű. Ezek körének meghatározásához ugyanis magukkal az adatszolgáltatókkal kell alapos konzultációkat folytatni. (Felmerülhet például olyan probléma is, hogy az adatszolgáltató a számítógéptől várja el két szám összeadását, ami adott esetben lehet megalapozott igény, máskor viszon nem.)

Az alapadatokat (és az ezeket karbantartó esetleges programokat) egyszer s mindenkorra eltárolhatjuk számítógépünk valamelyik adathordozóján; változtatásra csak a modell megváltozása esetén van szükség. Az elsődleges adatrendszer elemeit egy vagy több adathalmazban tárolhatjuk; ezek tartalma minden egyes menetrend esetén más és más lehet. A teljes körű adatszolgáltatás, valamint a gépi feldolgozás érdekében az elsődleges adatrendszert célszerű a számítógép által készített *adatlaprendszeren* kérni az adatszolgáltatótól. Ha az elsődleges adatrendszert már kialakítottuk, akkor ennek célszerűen csoportosított részhalmazait felvehetjük olyan adatlapokra, amelyek tartalmazzák a megfelelő *szöveges meghatározások*at, a modellhez szükséges esetleges *azonosító*kat, az input-folyamathoz szükséges esetleges *kódok*at, valamint az *adatelemek* helyét. A szöveges meghatározásokat, amik természetesen bármiféle információt tartalmazhatnak (például a megadandó adatelemek mértékegységét), az adatszolgáltató, az azonosítókat és az adatlapokra kerülő adatelemeket a programcsomag, míg a kódokat maga az input-rendszer használja fel. Az adatlaprendszert szintén állandó adatállományként kezelhetjük; megváltoztatására csak a modell, vagy az elsődleges adatrendszer határainak megváltozása esetén van szükség. (Valójában a nagy helyigény miatt nem érdemes azt az adatrendszert tárolni a számítógépben, ami adott esetben az adatszolgáltató előtt nyomtatott formában megjelenik; célszerű adatlapgyártó programokat is írni.)

A modellhez és a programcsomaghoz tartozó *automatizált input-rendszer*nek nevezzük egy adott modell és programcsomag esetében adathalmazok és programok olyan rendszerét, amely a fentebb vázolt formában tartalmaz

— alapadatokat;
— adatlaprendszert, amely teljeskörűen felöleli az elsődleges adatrendszert;
— algoritmusrendszert, amely teljeskörűen gondoskodik az előkészített adatrendszerről.

1.2 Adatszintek

Az előkészített adatrendszert változékonyság szempontjából általában jól elhatárolható *szintek*re oszthatjuk. Az egyik szintet nyilvánvalóan mindjárt az alapadatok alkotják, hiszen ezek, adott modell esetén állandók, változtatásukra nincs szükség: ezt nevezzük majd az előkészített adatrendszer nulladik szintjének. Definíciónk szerint viszont a nem alapadatok, bizonyos menetrendek esetén, mind megváltozhatnak, ezért ezek a pozitív szintekhez tartoznak. Az is világos, hogy az előkészített adatrendszer pozitív szintjeit visszavezethetjük az elsődleges adatrendszer (pozitív) szint-felosztására. Fontos itt megjegyeznünk, hogy míg az előkészített adatrendszer felosztása diszjunkt (azaz egy elem egy és csakis egy szinthez tartozhat), ez korántsem áll az elsődleges adatrendszerre; elképzelhető ugyanis, hogy egy-egy elsődleges adat hatással van mind kevéssé mind nagyon változékony előkészített adatokra is.

Az elsődleges adatrendszer logikailag összetartozó és egy szinten levő elemeit érdemes külön adathalmazban tárolni, és feldolgozásukat is külön algoritmussal (programmal) végezni. Az ilyen felosztás esetén az elsődleges adatrendszer különböző szinteken levő adathalmaz-csoportokból áll. Vegyük azonban észre azt is, hogy ily módon nem csupán egymástól független szintek jönnek létre, de ezen szintek között, legalábbis a bonyolultabb esetekben, bizonyos kapcsolatok is vannak. Itt most kétféle ilyen kapcsolatról szólunk. Egyrészt a feldolgozás *logikája* követel meg valamiféle sorrendiséget bizonyos adathalmazok között (például amikor az előkészített adatrendszer egy részét csak úgy tudjuk kialakítani az egyik adathalmazból, hogy közben fel kell használjuk egy másik adathalmaz feldolgozásának eredményét vagy részeredményét). Másrészt viszont törekednünk kell a számítástechnikai apparátus bizonyos részei (adatok tárolása, programok mérete, futási- és CPU-idők stb.) minél jobb kihasználására is; ez a feldolgozás sorrendjének — nyilvánvalóan csak a logikai után következő — *utilitárius* szempontját adja.

Ha két adathalmaz különböző szinten van, akkor érdemes szintjeik sorrendjében feldolgozni őket, hacsak az ilyen eljárás logikailag egyáltalán lehetséges. Ezzel szemben ugyanis utilitárius szempont nem szólhat, mellette viszont — nyilvánvalóan — igen. Az elsődleges adatrendszernek egy szintekre osztását *reguláris*nak nevezzük, ha a logikai szempont nem zárja ki, hogy a magasabb szinteken levő adathalmazok feldolgozása később következzék.

A logikai és utilitárius szempontok figyelembevétele alapján az elsődleges adatrendszer különböző szintjein levő adathalmazoknak egy speciális struktúráját kapjuk. (Ezt a struktúrát irányított gráffal is ábrázolhatjuk: az adathalmazok a gráf csúcspontjai; az A csúcspontból akkor mutat él a B csúcspontba, ha a fenti két szempont valamelyike miatt A feldolgozása megelőzi B-ét.) Ha a szintekre osztás reguláris, akkor az egyes szintek egymástól függetlenül kezelhetők; a szinteken belül érvényesülnek szempontjaink, s az egyes szintek sorrendjük szerint követik egymást.

1.3 Az input-kód

A legtöbb programcsomag esetében az előkészített adatrendszernek bizonyos mértékig kötött formátumúnak kell lennie; a leggyakoribb megkötés az input-adatok sorrendjére vonatkozik. Általában egyáltalán nem mindegy, hogy az előkészített adatrendszerben az adatok milyen sorrendben helyezkednek el. (Mi ugyan most nem foglalkozunk azzal az esettel, amikor a programcsomag nem követel meg valamiféle adat-sorrendiséget, annyit azért megjegyzünk, hogy nagy mennyiségű adat feldolgozása esetén még ilyenkor is célszerű valamilyen adat-sorrendet felállítani, már csak az áttekinthetőség és ellenőrizhetőség kedvéért is; az e pontban kifejtendő input-kódra ez esetben is szükség lehet.)

Az előkészített adatrendszer elemeinek a programcsomag által megkövetelt sorrendiségét *logikai sorrend*nek nevezzük. Ha pusztán a logikai sorrendet vennénk figyelembe, akkor — nagyszámú adat feldolgozása esetén — az esetek többségében a programcsomag által ugyan feldolgozható, ám az ember számára teljes mértékben áttekinthetetlen előkészített adatrendszert kapnánk. Ezt még akkor sem engedhetjük meg, ha mind az input-, mind az output-foyamatot teljes mértékben automatizáltuk (bizonyos elkerülhetetlen kézi ellenőrzések, hibakeresések miatt). Ezért szükség van az előkészített adatrendszer elemei *utilitárius sorrend*jének bevezetésére is: a programozónak (esetleg a felhasználóval közösen) meg kell állapítania azoknak az adatelemeknek a sorrendjét, amelyekről a programcsomag logikai sorrendet nem követel meg, hogy az adatrendszerben való eligazodás egyáltalán lehetséges legyen. (Vigyázzunk: az előző pontban nem adatelemeknek, hanem az adathalmazoknak és feldolgozásuknak logikai és utilitárius sorrendjéről volt szó.)

Nagy modellek esetén gyakorlatilag lehetetlen az alapadatokat és az elsődleges adatrendszert úgy feldolgozni, hogy a keletkező eredményhalmazokból az előkészített adatrendszer közvetlenül összeállítható legyen. (Gyakran történik meg ugyanis az, hogy összetartozó adatok más és más adathalmazok feldolgozása során keletkeznek.)

Ezért azután célszerű az előkészítet adatrendszer minden egyes különálló adatelemét egy ún. *input-kód*dal ellátni. (Ez nem szükségszerű: igen bonyolult programokkal el lehet érni a kívánt sorrendet az input-kód nélkül is; ez azonban hatalmas erőfeszítést igényel.) Az input-kódot tulajdonképpen speciális sorszámként foghatjuk fel, hiszen szerepe az adatelemek sorrendjének felállításában van, ám az input-kód nemcsak szám lehet, hiszen megfelelő kódolással a sorrendet másképp is kialakíthatjuk.

Az input-kódokat, illetve alkalmasan választott részeiket fel kell tüntetni az adatlapokon, így azok szerepelni fognak az elsődleges adatrendszer adathalmazaiban. A feldolgozó programoknak természetesen ezeket is kezelniük kell. (Ha például egy elsődleges adatból több előkészített adat is lesz, akkor ebből az egy kódból vagy kódrészletből kell kialakítani valamennyi előkészített adatkódját.)

Most már tisztán utilitárius szempontok alapján a programozó a feldolgozás folyamatába bizonyos pontokon beiktathat olyan lépéseket, amelyek egy vagy több feldolgozott adathalmaz eredményhalmazát az előkészített adatrendszerben elfoglalandó sorrendnek megfelelően alakítja ki és fűzi össze. Szó lehet egyes eredményhalmazok egyszerű rendezéséről, de több eredményhalmaz egyidejű rendezéséről és a szükséges sorrendnek megfelelő összefűzéséről is. Adott esetben elegendő lehet ezt a lépést az előkészített adatrendszer kialakításának utolsó

fázisában végrehajtani, amikor már az összes adathalmazt feldolgoztuk. Több pozitív, reguláris adatszint esetén azonban nagyon is célszerű az egyes szintek feldolgozásának lezárásaként beiktatni egy-egy ilyen lépést, hiszen a változékonyabb adatokkal történő újrafuttatáskor az alacsonyabb szintű adatokkal, illetve sorrendbe állításokkal, ilyen megoldás esetén, már nem kell foglalkoznunk.

1.4 A konstans adathalmaz

Az alapadatok, valamint a modell struktúrája alapján érdemes elkészíteni az input-folyamat ún. *konstans adathalmazát*. Ez egyrészt — a megfelelő sorrendben — tartalmazza az alapadatokat, másrészt alkalmasan megválasztott helyeken olyan információkat tartalmaz, amelyek alapján eldönthető az, hogy a sorba állított nem-alapadatokból az előkészített adatrendszer számára az adott ponton hány adatelemre van szükség. Ha rendelkezünk ilyen adathalmazzal, akkor az adatelőkészítés utolsó lépése a következőképpen alakul: Az input-kódok alapján elvégezzük az utolsó rendezést (összefűzést), majd a konstans adathalmaz első részlete egyfelől megmondja, hogy innen mely alapadatok kerüljenek az előkészített adatrendszer elejére (természetesen ez lehet üres részhalmaz is), másfelől pedig valamiképpen megmondja azt, hogy az imént összefűzött adathalmazból mely adatokra van szükség (természetesen most is lehet szó üres részhalmazról). Ezután a konstans adathalmaz második részlete egyfelől megmondja, hogy innen mely alapadatok kerüljenek az előkészített adatrendszer következő helyeire, másfelől pedig ugyancsak az összefűzött adathalmazból szükséges soron következő adatokat határozza meg; és így tovább.

A konstans adathalmaz, mint a neve is mutatja, állandó (legalábbis változatlan modell esetén). Jelentősége azonban nemcsak az ismertetett eljárás célszerűségében áll, és nemcsak ezért kezeljük kiemelten, hanem ellenőrzési szerepe miatt is. Elképzelhető lenne az is, hogy ugyanúgy kezeljük, mint a többi adathalmazt, azzal a különbséggel, hogy a nulladik szinthez tartozónak tekintjük; elemeit input-kóddal látjuk el, és a feldolgozás első fázisában vesz részt.

A konstans adathalmaz tartalmazza a modell vázát, s mint ilyet, a modell ismert matematikai leírása segítségével, egyszer s mindenkorra összeállíthatjuk. A benne tárolt információk alapján ellenőrizhetjük, hogy az előkészített adatrendszer a megfelelő helyeken valóban annyi adatelemet tartalmaz-e, amennyit kell. Ennek óriási a jelentősége, még ha ez a funkció nem is tesz feleslegessé számos egyéb fontos ellenőrzési feladatot.

(A konstans adathalmaz alapadatait természetesen elláthatjuk a megfelelő input-kóddal. Ennek célja azonban nem az, hogy belőle kiindulva kapjuk meg az előkészített adatrendszert ugyanolyan összefűzéssel, ahogy a pozitív szintekhez tartozó feldolgozott adathalmazokat kezeljük, hanem egyrészt további ellenőrzések céljára szolgál, másrészt pedig az előkészített adatrendszert egységesíti.)

1.5 Több lépcsős input-folyamat

Ha a programcsomag rugalmasan kezelhető saját adattárolási és karbantartási lehetőségekkel rendelkezik, akkor a modellt nem szükséges egy lépcsőben felépíteni. Ilyen esetben lehetséges az is, hogy csak a modell egy (nagy) részé-

nek előkészített adatrendszerét állítjuk elő az első lépcsőben, majd — a programcsomag karbantartó részeit használva — a további lépcsőkben további előkészített adatrendszereket készítünk. Különösen a felső szintek feldolgozását célszerű a későbbi lépcsőkben elvégezni, hiszen az igen változékony adatokkal amúgy is sokszor kell módosítani a modell adatrendszerét, s az ehhez szükséges munka nagy részét a programcsomag készítői már elvégezték.

1.6 Az automatizált input-rendszer működésének összefoglalása

A) Rendelkezésünkre áll az 1.1 pontban leírt adatlaprendszer.

B) Rendelkezésünkre áll az 1.4 pontban leírt konstans adathalmaz.

C) Rendelkezésünkre áll az 1.1 pontban leírt elsődleges adatrendszernek az 1.2 pontban leírt (esetleg reguláris) szintekre osztása.

D) Esetleg rendelkezésünkre áll az input-folyamat 1.5 pontban leírt lépcsőkre bontása.

E) Rendelkezésünkre áll az 1.1 pontban leírt algoritmusrendszer.

1. A megfelelő programokkal kinyomatjuk az adatlapokat (a dokumentálás végett több példányos papírra !).

2. A kitöltött adatlapokról a releváns adatokat betöltjük az elsődleges adatrendszer különböző szintjein levő adathalmazokba.

3. A figyelembe vett logikai és utilitárius szempontok alapján elvégezzük az egyes adathalmazok megfelelő sorrendű feldolgozását, az alkalmasan megválasztott helyeken közbeiktatva egyes adathalmazok rendezését, illetve összefűzését.

4. A konstans adathalmaz és az utoljára összefűzött adathalmaz segítségével előállítjuk az előkészített adatrendszert. (Több lépcsős input-folyamat esetén az első előkészített adatrendszert.)

5. Használjuk a programcsomagot.

6. Több lépcsős input-folyamat esetén a programcsomag karbantartó része számára elkészítjük a további előkészített adatrendszereket.

7. Használjuk a programcsomagot.

Megjegyzések

1. A konstans adathalmazzal kapcsolatos kiemelt adatellenőrző lépést leszámítva nem említettünk eddig semmiféle más, az input-folyamatban szereplő ellenőrzési tevékenységet — bár ilyenre nyilvánvalóan igen nagy szükség van. Úgy gondoltuk azonban, hogy efféle tevékenységet minden programban kell végezni; a tisztán ellenőrzési funkciót betöltő programokat pedig az algoritmusrendszer részének tekintettük.

2. A fentiekben egy általános sémát ismertettünk. Ez természetesen nem jelenti azt, hogy minden konkrét esetben mereven ragaszkodni kellene hozzá; mindössze egy olyan gondolati vonulatot kívántunk felvázolni, ami az egyes esetekben a kivitelezéshez támpontokat nyújt. Mint látni fogjuk, az általunk megvalósított konkrét automatizált input-rendszer is mutat eltéréseket az általános sémától.

2.1 A konkrét programcsomag és modell rövid ismertetése

A Magyar Tudományos Akadémia Matematikai Kutató Intézetének munkatársai hosszabb ideje vesznek részt a Magyar Alumíniumipari Tröszt (MAT) hosszú távú stratégiai tervezési feladatát segítő nagyméretű modell kidolgozásában, felállításában és vizsgálatában.

A modell tulajdonképpen egy vegyes-egészértékű lineáris programozási feladat. Ezt a MAT IBM 4331 típusú számítógépén megtalálható MPSX-MIP/370 (*M*athematical *P*rogramming *S*ystem *E*xtended — *M*ixed *I*nteger *P*rogramming) programcsomag segítségével oldjuk meg.

Az MPSX—MIP/370 [1] igen alkalmas igen nagyméretű (akár 16 ezer feltételt tartalmazó) vegyes-egészértékű lineáris programozási feladatok megoldására. Nem célunk itt a programcsomag általános ismertetése, mindössze annak input-rendszeréről mondjuk el a témánkhoz illeszkedő legfontosabb részleteket.

A programcsomag az input-adatokat háromféle formában képes feldolgozni. Mi a modell mérete, az ellenőrzési lehetőségek, és az input-folyamat könnyebb automatizálhatósága érdekében azt választottuk, amelyik az adatokat rögzített formátumban, kártyaképek alakjában várja. Az egyes kártyaképek vagy indikátorok (ezek bizonyos adatcsoportok jelzésére, elválasztására szolgálnak, s így számadatokat nem is tartalmaznak), vagy pedig adatsorok. Az adatsorokat az MPSX-MIP/370 hét adatmezőre osztja. Az első egy indikátor-kód (ezzel adjuk meg például, hogy az egyes feltételeket korlátozzuk-e, és ha igen, akkor alsó, illetve felső korlátról, vagy pedig egyenlőségről van-e szó); a második, valamint a harmadik és az ötödik (utóbbi kettő lehet üres is) oszlop- vagy sorazonosító (ezek felelnek meg a mátrix változóinak, illetve feltételeinek; az MPSX-MIP/370 legtöbb outputja ezeken a neveken hivatkozik a megfelelő változókra, illetve feltételekre); a negyedik és a hatodik (mindkettő lehet üres is) tartalmazza a megfelelő számadatot; végül a hetedik, amelyet az MPSX-MIP/370 adatfeldolgozója tulajdonképpen figyelmen kívül hagy, teremti meg az input-kód bevezetésének lehetőségét. Az egyes mezők meghatározott pozíciókon kezdődnek és végződnek.

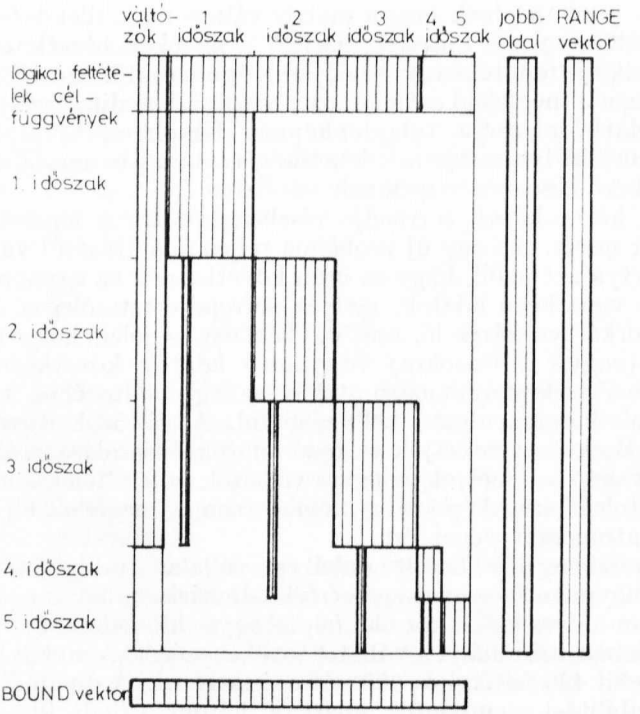
Az egyes kártyaképek sorrendje részben kötött. A legelső adja meg az adatrendszer nevét. Ha egy új probléma teljes felállításáról van szó, akkor a második kártya azt jelöli, hogy ez után következnek az egyes sorokra (vagyis feltételekre) vonatkozó adatok, melyek sorrendje tetszőleges. Ezt a blokkot egy indikátorkártya zárja le, ami egyben azt is jelzi, hogy most az egyes oszlopokra (vagyis változókra) vonatkozó adatok következnek. Ezeket a kártyaképeket oszlopfolytonosan, tehát az egy változóhoz tartozó adatok egymás után, hézagmentesen kell megadni. A változók sorrendje azonban tetszőleges. Ezek és a hozzájuk tartozó adatok felsorolása után következnek azok a kártyaképek, amelyek az egyes változók és feltételek korlátozását szolgálják. Az utolsó kártyakép jelzi a programcsomag megfelelő eljárása számára, hogy az adatrendszer véget ért.

A programcsomag által kezelt modell egy vállalat stratégiai tervezési céljaira alkalmas, nagyméretű vegyes-egészértékű lineáris programozási feladat. A modell három ötéves időszakot ölel fel; az egyes időszakokban, meghatározott szintű bontásban, szimulálja a vállalat tevékenységét. A mátrixban időszakonként körülbelül 400 feltétel és 600 változó szerepel. A dinamikára vonatkozó bizonyos stabilitási szempontok miatt a három valódi időszakhoz hozzávettünk még két, összesen mintegy 30 feltételt és 50 változót tartalmazó

pszeudo-időszakot. Így a mátrixot körülbelül 1200 feltétel, 1800 változó és 10 000 nem nulla elem alkotja. Az egyes időszakok közötti összefüggés szűk keresztmetszeten jelentkezik, ezért a mátrix lényegében három diagonálisan elhelyezkedő almátrixból áll.

A stratégiai tervezést 12 fejlesztési körhöz kapcsolódóan 31 egész-változó segítségével írjuk le. Egy-egy fejlesztési körhöz 2—4 egész-változó tartozik, amelyek mindegyike csak a 0 vagy az 1 értéket veheti fel. Az azonos fejlesztési körhöz tartozó változók összege kötelezően 1. Ez azt jelenti, hogy a feladat megengedett megoldása esetén minden egyes fejlesztési körben pontosan egy egész-változó értéke egyenlő 1-gyel, az összes többié 0-val. Ezek az egész-változók képviselik tehát a döntéseket. Minden fejlesztési kör a vállalat egy nagyobb részterületének felel meg. Az egyes egész-változók reprezentálják az erre a területre vonatkozó, egymástól eltérő fejlesztési pályákat, amelyek közül pontosan egynek kell megvalósulnia. Amennyiben egy egész-változó értéke a megoldásban 1, úgy a hozzá kapcsolódó beruházások, kapacitások, egyéb ráfordítások (munkaerő, munkabér, karbantartás stb.) „élnek", az ehhez a fejlesztési körhöz tartozó többi egész-változó által meghatározott adatok nem játszanak szerepet a megoldás többi részének kialakításában. Az egész-változókra vonatkozó adatok természetesen mind a három tervidőszak tevékenységét meghatározzák, így az ezekhez a változókhoz tartozó mátrix-elemek nem korlátozódnak egy almátrixra.

A modell szerkezetét az 1. ábra szemlélteti.



1. ábra

2.2 A modell adatrendszere

A programcsomagnak körülbelül 10 000 számadatra van szüksége ahhoz, hogy a modellt a számítógépen felállítsuk. Ezen adatmennyiség mintegy harmada konstans (általában + 1 vagy — 1): ezek alkotják a nulladik szintű adathalmazt, az alapadatokat.

A többi adatot három szintre osztottuk be. Az első szinten azok a fajlagosok vannak, amelyek a vállalaton belüli technológiai folyamatok leírásához szükségesek. A második szintre kerültek az egyes döntési (egész) változókhoz kapcsolódó adatok (kapacitások, a szükséges beruházások, ezek felosztása, az ezekhez kapcsolódó egyéb költségek, adók stb.). Végül a harmadik, legmagasabb szinten a vállalat által vásárolt, illetve eladott termékek különféle árai (kétféle tőkés export, tőkés import, szocialista export és import, valamint belföldi) és a hazai ellátási feladatokat szabályozó adatok vannak. Az első és a második szintet azért érdemes elkülöníteni, mert a technológiai folyamatok fajlagosai új fejlesztési pályarendszer (azaz a második szint) megadása hiányában nem változnak, míg olyan új fejlesztési változatok bevezetése, amely a fajlagosokat nem módosítja, nagyon is elképzelhetők.

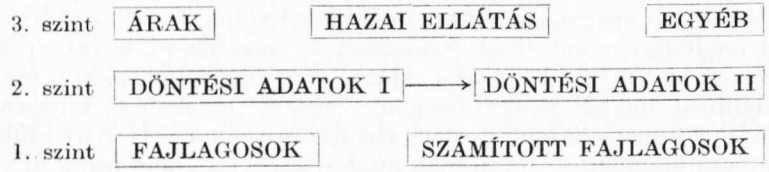
Az elsődleges adatrendszer első szintjén két adathalmaz van. Az egyik azokból a fajlagosokból áll, amelyeket változtatás nélkül írhatunk be a mátrix egyetlen pozíciójába. A másik adathalmazba az olyan elsődleges adatok kerültek, amelyeket vagy nem lehet közvetlenül beírni a mátrixba, mert bizonyos mértékű feldolgozásra szorulnak, vagy pedig a mátrix két vagy több pozíciójába kell beírni őket.

A második szinten ugyancsak két adathalmaz található. Az egyik feldolgozásának mellékterméke ként olyan közbülső adathalmazt kapunk, amelyet a másikkal együtt lehet földolgozni.

Végül a harmadik szinten három adathalmaz van: az elsőben az árak, a másodikban a hazai ellátási feladatokkal kapcsolatos adatok, a harmadikban pedig néhány olyan vegyes adat, amit logikailag máshová nem lehetett besorolni.

Ily módon az elsődleges adatrendszer reguláris szintekre osztását kapjuk. Ezt a 2. ábrán szemléltetjük.

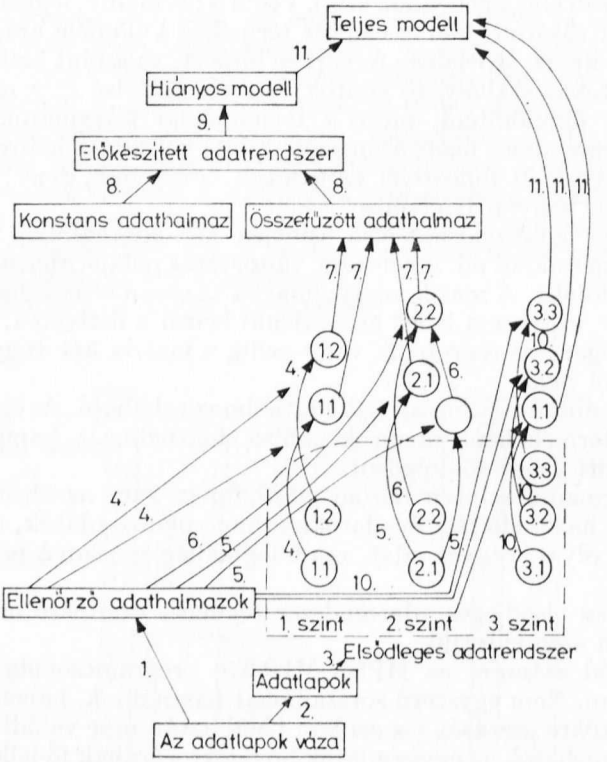
Az input-kód számára az MPSX-MIP/370 programcsomag 9 karakternyi helyet tart fenn. Nem egyszerű sorszámokat használunk, hanem a 9 karakternek olyan mezőkre osztását (és azokon belül aztán már valódi sorszámozást), amelyből kiolvashatók az egyes adatok hovatartozásának fő jellemzői (például: melyik időszakról, melyik változócsoportról, változóról, azon belül melyik adatelemről van szó stb.). Így akár az input-kódból, akár a (programcsomag által használt) azonosítóból rögtön leolvasható, hogy egy változót, például a vállalat melyik egységének tevékenységéhez használunk.



2. ábra

2.3 A modell algoritmusrendszere

Az input-folyamat két lépcsős. Az első lépcsőben dolgozzuk fel a nulladik, az első és a második szinten levő adatokat, majd a programcsomag problémafelállító eljárása segítségével a modellt — még hiányosan — felvisszük a gépre és el is tároljuk. A második lépcsőben a harmadik szinten levő adatokat dolgozzuk fel, és az MPSX-MIP/370 egyik módosító eljárása segítségével az előzőleg eltárolt modellt megváltoztatjuk, a hiányzó adatokkal kipótoljuk, és az immár teljessé vált modellt tároljuk el.



3. ábra

Esetünkben az első szinten levő adathalmazok feldolgozása eredményeként kapott adathalmazoknak az input-kód szerinti rendezése, illetve összefűzése igen rövid futási idejű programot eredményezett volna. Ezért az első és a második szint eredményhalmazainak rendezését és összefűzését egyetlen programmal végezzük el. Végül a harmadik szinten levő adathalmazokat a már részlegesen felállított modell módosításaként vesszük figyelembe, kihasználva az MPSX-MIP/370 speciális lehetőségeit. Rendezésre ekkor már nincs szükségünk, mert a programcsomag — lévén csak módosításról szó, nem pedig új vektorok hozzáadásáról — az adatokat „magától" a helyükre teszi. Ez a módszer azért

is hasznos, mert hasonló jellegű módosításokat (például új árrendszer esetén) a teljesen felállított modellel is gyakran kell végezni.

Adatlapjainkat külön programok állítják elő. Az adatlapok vázát lemezen tároljuk. Ebből készülnek egyrészt maguk a nyomtatott adatlapok, másrészt az egyes adathalmazok feldolgozásának ellenőrző adathalmazai. Az ellenőrzés úgy történik, hogy az ellenőrző adathalmaz egyes adatait egybevetjük az adathordozóra felvitt kitöltött adatlapokkal. Ily módon az adatfelvitel során az azonosítókban vagy az input-kódokban bekövetkező esetleges hibákat igen nagy valószínűséggel tudjuk kiszűrni.

Végeredményben tehát a következő sémát kapjuk (l. a 3. ábrát):

1. Az adatlapok vázából ellenőrző adathalmazok készítése.
2. Az adatlapok vázából adatlapok készítése.
3. A kitöltött adatlapok felvitele.
4. Az első szint két adathalmazának feldolgozása.
5. A második szint első adathalmazának feldolgozása; egyszersmind input készítése e szint másik adathalmazának feldolgozásához.
6. A második szint másik adathalmazának feldolgozása.
7. Az eddig kapott négy eredmény-adathalmaz rendezése és összefűzése.
8. Az előbb kapott és a konstans adathalmaz összefűzése.
9. A modell hiányos felállítása.
10. A harmadik szint három adathalmazának feldolgozása.
11. Az előbbiek alapján a már felállított modell módosítása, azaz teljessé tétele.

2.4 Tapasztalatok az automatizált input-rendszerrel

A fent jelzett programok a MAT IBM 4331 típusú számítógépre vannak telepítve. A forrásprogramok nyelve az MPSX-MIP/370 által használt ún. ECL (*E*xtended *C*ontrol *L*anguage) nyelvvel kompatibilis PLIOPT. A megfelelő program-könyvtárakban minden egyes fent jelzett (l. a 3. ábrát) program megtalálható forrásnyelvi, lefordított és megszerkesztett (futtatható) és modul (azaz lefordított és más programba beszerkeszthető) formában. Létezik továbbá egy olyan, paraméterekkel irányítható program is, ami — kérés esetén — ezek közül a modulok közül választ; ily módon egyetlen programmal futtathatjuk akár a teljes input-folyamatot, akár annak bármely részletét.

A jelentősebb lépések a CPU-idő felhasználása szempontjából a következők (a 3. ábra jelöléseivel):

5. A DÖNTÉSI ADATOK I feldolgozása
7. A négy eredményhalmaz rendezése és összefűzése
8. Az előkészített adatrendszer elkészítése
9. A hiányos modell felállítása
11. A teljes modell felállítása.

Ezek közül is kiemelkedik a 7. lépés (rendezés-összefűzés); ez a teljes input-folyamat CPU-idő szükségletének körülbelül 90 %-át fogyasztja el. Ez azonban csak akkor van így, ha a teljes input-folyamatot futtatjuk. Ha ugyanis az 5. programlépést (a DÖNTÉSI ADATOK I feldolgozása) kihagyjuk, akkor a 7. lépés CPU-idő igénye (ha egyáltalán szükség van erre a lépésre az adott feldolgozási eljárás során) a teljes folyamat CPU-idő igényének csak mintegy harmada lesz.

Látható tehát, hogy eme speciális input-rendszer sarkpontját a döntési változókhoz kapcsolódó nagymennyiségű adat feldolgozása, és a feldolgozott adatok rendszerbe illesztése jelenti. Ez nyilvánvalóan a modell — stratégiai — jellegéből fakad.

Nem állítjuk természetesen azt, hogy a fent vázlatosan leírt automatizált input-rendszer létrehozásával és működtetésével az adatfeldolgozás és az adatelőkészítés valamennyi problémáját megoldottuk. Különösen akkor nem mondhatjuk ezt el, ha valamiféle optimumot szeretnénk elérni. Ennek ellenére úgy véljük, hogy mindazokat a lehetőségeket, amiket — e feladatmegoldásához — a számítógép kínál, nagy mértékben ki tudjuk használni a fenti séma alkalmazásával.

A fennmaradó problémák természete tulajdonképpen szervezési: egy működőképes automatizált input-rendszer olajozott használhatósága a szervezési feladatok hatékony megoldásától függ, hiszen az elsődleges adatok körében az adatszolgáltatás csak a géptől függetlenül, a modellt használó szervezet megmozgatásával érhető el.

IRODALOM

1. Hanák G.: Az MPSX-MIP/370 matematikai programozási programcsomag. Néhány konkrét tapasztalat. *Szigma*, 1984. XVII/4, 267—281.