3. TÖMBÖK

Az egyszerű adattípusok ismertetését a tömbökkel kezdjük. Az sem okozna zavart, ha ezt a fejezet nem szerepelne jegyzetünkben, tekintettel arra, hogy a tömböket ismerjük a programozási kurzusokból. Mégis, célszerű néhány jellemző megállapítást tenni ezzel az alapvető struktúrával kapcsolatban.

- A tömbök legfontosabb tulajdonsága az, hogy elemei – indexeléssel – közvetlenül elérhetők. Ezt ADT szintű tulajdonságnak tekinthetjük.
- A tömbről ADS szinten „tömbszerű" képünk van, de az elemek rákövetkezősége alapján irányított gráfként is felfoghatunk egy tömböt. Ez az absztrakció a láncolt ábrázolás felé mutat.
- Az megvalósítás során, az ADS szinttel összhangban, ritkán választjuk a tömb láncolt ábrázolását (ahogyan- fordítva – a listák esetében sem gyakori a tömbös megvalósítás).
- A tömbökről nem könnyű megmondani a felhasználás egy körében vagy konkrét esetében, hogy saját műveletekkel rendelkező önálló típusnak, vagy csupán a reprezentáció adatszerkezetének tekinthetők.
- A tömbök dimenziószámmal rendelkeznek; a vektor egydimenziós, míg a mátrix kétdimenziós ismert struktúra; de magasabb dimenziós tömbök használata sem ritka. A tömb erősen szemléletes fogalom; három dimenzióig könnyű elképzelni, lerajzolni a szerkezetüket.
- Ma már egy többdimenziós tömb, például egy mátrix nem jutattja eszünkbe, hogy a további lépésként egydimenziós tömbbel kellene reprezentálni. Ez annak köszönhető, hogy a programozási nyelvek elemi lehetőségként kínálják a többdimenziós tömbök használatát. Érdemes azonban tudatosítani, hogy többdimenziós tömbök a számítógép memóriában egydimenzióként ábrázolódnak.
- Speciális több dimenziós tömbök (például alsóháromszög-mátrix) helytekerékos egydimenziós ábrázolásáról olyakor magunk gondoskodunk.

Ezeket a gondolatokat valamivel részletesebben is kifejtjük a következő pontokban.

3.1. A tömb absztrakt adattípus

Legyen T az E alaptípus feletti k (≥ 1) dimenziós tömb típus. Vezessük be az $I = I_1 \times ... \times I_k$ indexhalmazt, ahol $I_j = [1 .. n_j]$ $(1 \leq j \leq k)$. (Megjegyezzük, hogy az indexelés 1 helyett kezdődhetne általában mj-vel is, de az egyszerűség kedvéért 1-et fogunk használni.)

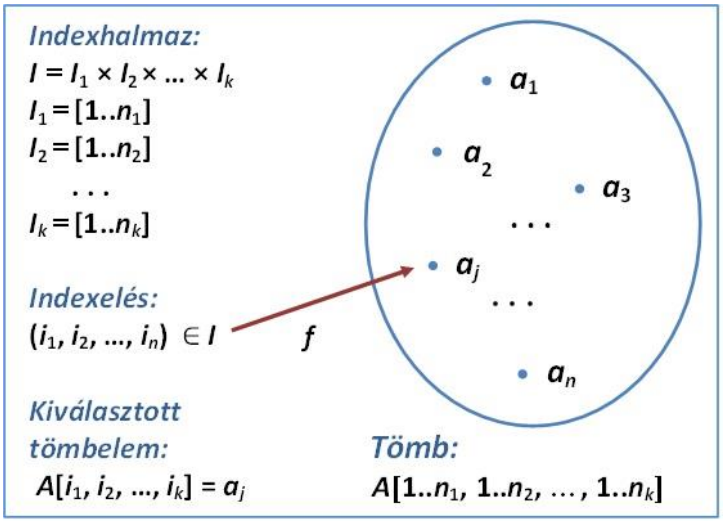
Az $A \in T$ tömbnek így $n = n_1 \cdot n_2 \cdot ... \cdot n_k$ elemet tárol, amelyek halmazát $\{a_1,...,a_n\}$ jelöli. A T tömbhöz tartozik, mint a típus meghatározó komponense, egy $f : I \mapsto [1..n]$ indexfüggvény, amely kölcsönösen egyértelmű leképezést létesít az indexhalmaz és a elemek halmazbeli indexei között, ezáltal egyértelmű leképzést valósít meg az indexhalmaz és a tömb elemei között. A tömbelemek egyértelműen és közvetlenül elérhetővé válnak az indexfüggvény alkalmazásával.

Bevezetjük az A [1 .. n_1, 1 .. n_2, ..., 1 .. n_k] jelölést az A tömbre, amely magában foglalja az indexhalmazát és utal arra, hogy az indexkifejezések és a tömelemek közötti kapcsolat is adott, így annak alapján az elemekre – indexeléssel – lehet közvetlenül hivatkozni.

Bevezetjük az $A[i_1, i_2, ..., i_k]$ jelölést a tömbelemek indexelésére. Ha a fenti indexfüggvény szerint $f(i_1, i_2, ..., i_k) = j$, akkor ez az indexelés az a_j elemet választja ki

$$A[i_1, i_2, ..., i_k] = a_j$$

Az indexelés mechanizmusát (absztrakt megközelítésben) a 3.1. ábra szemlélteti.



3.1. ábra. Tömb absztrakt adattípus

A tömb műveleteinek köre szerény: a most bevezetett indexeléssel lekérdezhetjük a tömb elemeit, emellett módosíthatjuk is azokat. A tömb a mérete nem változik; nem lehet a tömbbe egy új elemet beszúrni, és nem lehet a tömbből egy elemet kitörölni.

A szokás elnevezések szerint $k = 1$ esetén a vektorról, $k = 2$ esetén a mátrixról beszélünk.

3.2. A tömb absztrakt adatszerkezet

A tömböktől elválaszthatalan a szerkezetükről alkotott kép, amely jórészt a matematikai tanulmányaink során alakult ki. Ezen a kép alapján például egy cellákból álló lineáris vagy „négyzethálós” sémában helyezzük el (az egy, illetve kétdimenziós) tömb elemeit. A 3.2. ábrán egy mátrix szokásos ábrája látható.

A [1..3, 1..4] =

10	-30	70	100
-50	120	40	-20
-80	60	110	-90

3.2. ábra. Tömb szemléletes képe (ADS)

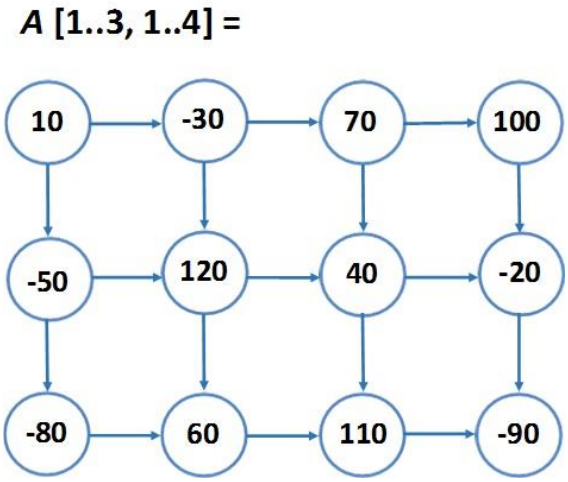
Az absztrakt adatszerkezet bevezetésével a fenti f indexfüggvény a háttérbe húzódik, hiszen azt lehet mondani például a fenti kétdimenziós tömb esetén, hogy mondjuk az $A[2, 3] = 40$ elem a 2. sor 3. eleme, vagyis az indexkifejezést vizuálisan megjelenítettük az ADS szintű sémával.

Szemléletünk számára tehát a vektor egy beosztásokkal ellátott szalag, a 2-dimenziós tömb egy mátrix, a 3-dimenziós tömb egy cellákra osztott téglatest alakját ölti gondolatainkban.

Az előző, bevezető jellegű 2. fejezet szerint, az absztrakt adatszerkezetet általában egy olyan irányított gráf szemlélteti, amelyben az élek az adatelemek közötti rákövetkezéseket jelenítik meg. Egy *k*-dimenziós tömb elemeinek általában, a dimenziók határát kivéve, *k* számú rákövetkezőjük van. Formálisan isbevezethetjük a *j* szerinti rákövetkezés fogalmát:

$$köv_j\, A\,[i_1,...,i_j,...,i_k\,] = A\,[i_1,...,i_{j+1},...,i_k\,] \qquad (i_j < n_j)$$

A 3.3. ábra egy kétdimenziós tömb, az $A\,[1..3,\,1..4]$ mátrix absztrakt gráfszerkezetét mutatja. Ez egy olyan ortogonális struktúra, amelyben minden csúcsból két él vezet a tömbbeli rákövetkezőkhöz.



3.3. ábra. Tömb gráfja (ADS)

Valójában a tömbről nem ilyen képet őrzünk fejünkben, ahogyan a verem sem egy lineáris gráf formájában rögzül a memóriánkban. Annyiban azonban mégsem fölösleges a gráfos szemlélet, mert közelebb hozza a ritka mátrixok láncolt ábrázolásának ötletét.

3.3. A tömb megvalósításai

Ebben a pontban két ritkán felmerülő kérdést érintünk. Az egyik a többdimenziós tömbök egydimenziós ábrázolására vonatkozik. A magasszintű programnyelvek elfedik előlünk ennek szükségességét, azonban hasznos lehet megismerni a fordítóprogramokba épített tárolási elvet. A másik kérdés a nagyméretű, de kevés adatot tartalmazó tömbök helytakarékos tárolására vonatkozik. Bemutatjuk a ritka kitöltöttségű mátrixok láncolt ábrázolásának módszerét.

3.3.1. Aritmetikai reprezentáció

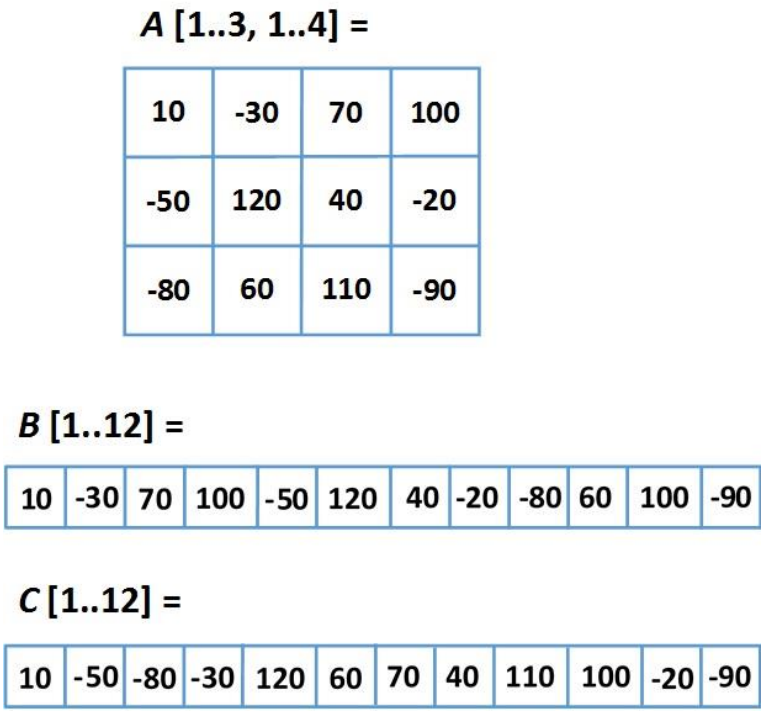
Egy adatszerkezet aritmetikai ábrázolása során az adatelemeket egy tömbben helyezzük el, az eredti strukturális összefüggéseket pedig függvények formájában adjuk meg.

Az adatokat tároló tömb lehet egy- vagy többdimenziós. Szemléletünk számára egy többdimenziós tömb már annyira egyszerű adatszerkezet, hogy nem szükséges mindig újra meggondolni az egydimenziós elhelyezés lehetőségét. A programnyelvek is megerősítenek ebben, hiszen a többdimenziós tömbök használatát az alapvető lehetőségek között nyújtják.

A tömb adattípus ismertetésekor azonban, legalább ezen a helyen egyszer, érdemes szóba hozni azt, hogy a többdimenziós tömbök elemeit még el kell helyezni a szalagszerű egydimenziós memóriában.

A szekvenciális tárolást általában a sorfolytosnos vagy oszlopfolytonos módszerrel szokás megoldani. (Azzal még itt sem foglalkozunk, hogy a tárolás végállomása egy bájtokból álló vektor, és a bájtokat – még tovább finomítva – bitek sorozata alkotja.)

A 3.4. ábrán a korábban is szereplő mátrixnak az egydimenziós tárolást illusztrálja, mindkét elhelyezési stratégia szerint.



3.4. ábra. Kétdimenziós tömb egydimenziós tárolása

A kapcsolatot az elemek mátrixban elfoglalt pozíciója és az újy helye között index-függvényekkel adjuk meg. Egy $A[1..m, 1..n]$ kétdimenziós tömbre, például a sorfolytonos esetben ez a következő:

$$ind(i,j) = (i-1)n + j$$

ahol $1 \leq i \leq m$ és $1 \leq j \leq n$. Esetünkben például, ha az $A[2,3] = 40$ elemet keressük a sorfolytonosan kialakított B tömbben, akkor azt a $(2-1)4 + 3 = 7$ indexű helyen találjuk: $B[7] = 40$.

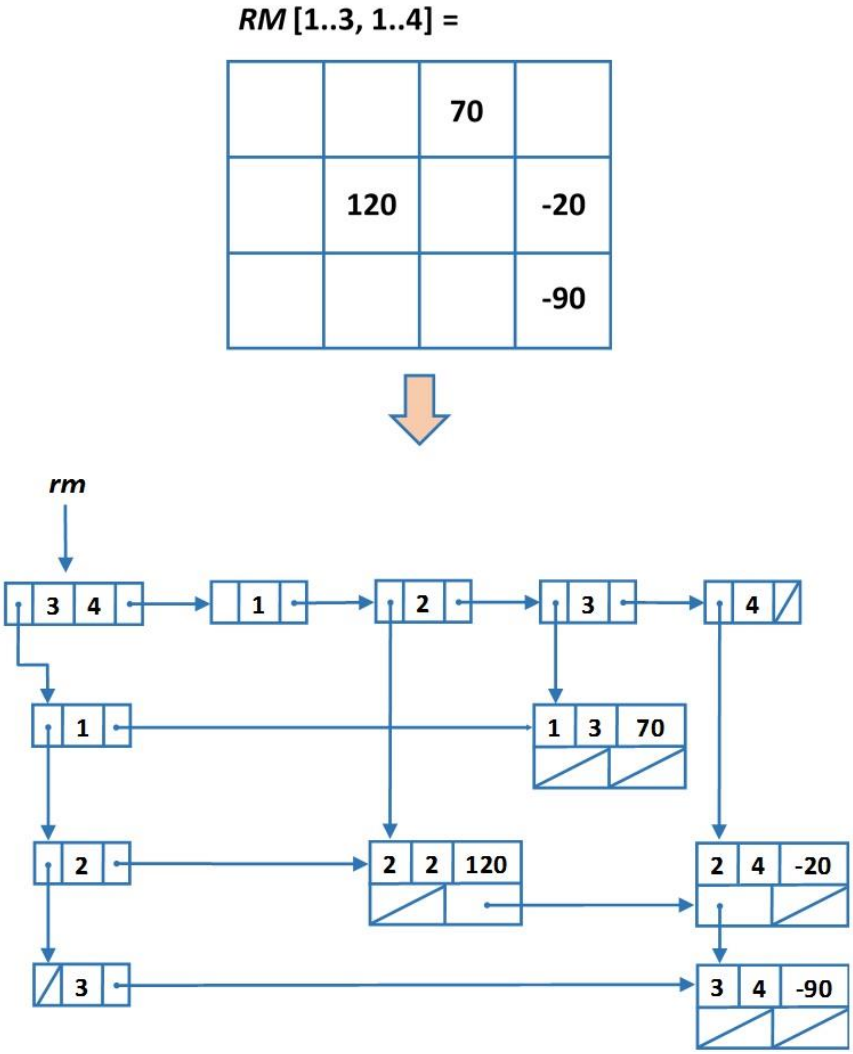
Ezzel a kérdéssel a további fejezetekben nem foglalkozunk. Ezután már egy többdimenziós tömböt is az ábrázolás eszközének tekintünk.

3.3.2. Láncolt ábrázolás

Bizonyos (jobbára gazdasági) problémák modellezése nagyméretű mátrixok alkalmazásához vezet. Előfordul azonban, hogy a mátrix csekély értékes adatot tárol. Ilyenkor gondolhatunk arra a reprezentálási módra, amelyet a 3.5. ábrán láthatunk.

A ritka mátrixok láncolt ábrázolásában csak az értékes elemek szerepelnek. Alkalmazhatunk egyirányú láncolást, amelyben minden elemtől a sor- és oszlopbeli rákövetkezőjéhez irányít a két tárolt pointer. A sorok és az oszlopok „bejárataira”,

pontosabban az első bennük szereplő elemre, egy-egy fejelem mutat, amelyek maguk is egy-egy litát alkotnak. A két fejelem lista egy közös fejelemből érhető el.



3.5. ábra. Ritka mátrix láncolt ábrázolása

Az ábra nem csak a helytakarékosság lehetőségét érzékelteti, hanem azt is, hogy ezzel az ábrázolási móddal feladtuk az elemek közvetlen indexelhetőségét és csak meglehetős nehézkességgel tudunk eljutni az elemekhez.

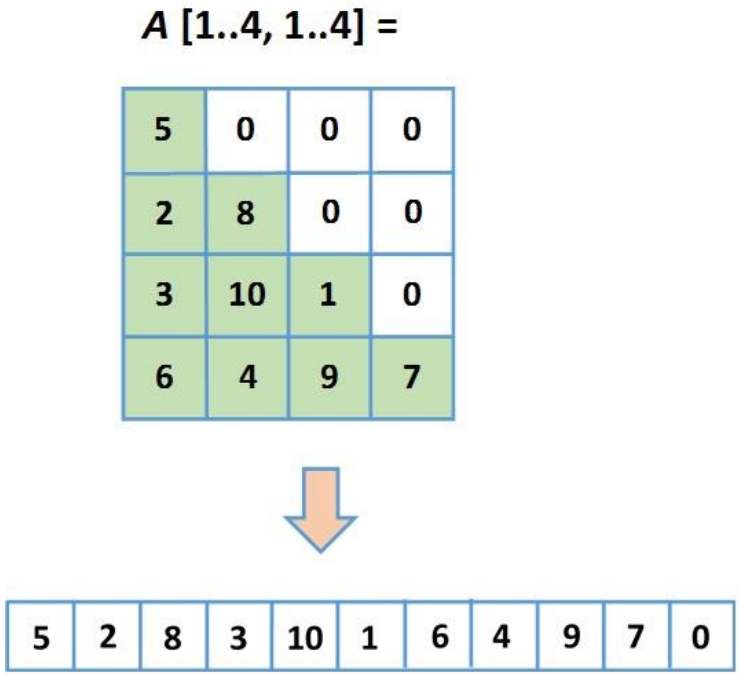
Az elemek elérését, illetve a struktúrában való mozgást valamelyest javítja, ha kétirányú láncolást alkalmazunk, mind a sorokban és az oszlopokban, mind pedig a két fejelem-listában.

Azt a kérdést, hogy alkalmazzuk-e adott esetben ezt a tárolási formát, két szempont dönti el. Egyik a helytakarékosság kérdése: az értékes mátrixelemek (várható) számának és méretének ismerete esetén könnyen kiszámítható, hogy előnyösebb-e ez az ábrázolás, mint a hagyományos. Ehhez csak a pointerek számát kell meghatározni, amelyet a memóriacím helyfoglalásával (ami általában 2 vagy 4 bájt) szorozva kell a memóriaigény számításában figyelembe venni.

A másik mérlegelendő szempont az, hogy mennyire támogatják a feldolgozó eljárások megvalósíthatóságát a láncolt adatszerkezeten való közlekedés korlátozott lehetőségei. A mai memóriakapacitások mellett lehet, hogy ez a súlyosabb szempont. Ha arra gondolunk például, hogy hogyan kellene két ritka mátrix összegét előállítani, akkor látható, hogy a láncolt ábrázolás mellett a legegyszerűbb feladadása megoldása is körülményessé válhat.

3.4. Speciális tömbök

A tömbök egydimenziós ábrázolásával azért sem fölösleges megismerkedni, mert találkozhatunk olyan adatszerkezetekkel, amelyeknél azt magunk valósítjuk meg. Ezek az adatszerkezetek általában olyan mátrixok, amelyeknek jelentősen kisebb helyen is tárolhatók, például azért, mert az elemek jó része hiányzik, és a hiány, ezzel együtt értékes rész is szabályos tartományt képez.



3.6. ábra. Alsó-háromszögmátrix sorfolytonos ábrázolása

A 3.6. ábrán látható alsóháromszög-mátrixban a főátló fölötti elemek hiányoznak. Az értékes elemeket sorfolytonos módon elhelyezve egy egydimenziós tömbbe memória-takarékos tárolást valósítunk meg, hiszen így a helyfoglalás közelítőleg a felére csökken.

Egy $n \times n$-es négyzetes mátrix esetén az alsó háromszög tárolásához szükséges cellák száma: $n(n+1)/2+1$. Megállapodás szerint, az értékes elemek után még egy nullát is elhelyezünk.

Ez az átpakolási stratégia az alábbi index-függvénnyel válik követhetővé:

$$ind(i,j) = \begin{cases} (i-1)i/2 + j, & ha\ 1 \le i \le j \le n \\ n(n+1)/2 + 1 & \text{egyébként} \end{cases}$$

(A nulla elem tárolása egy általánosabb eljárás következménye. Ha ugyanis a mátrixban lenne néhány olyan elem, amelyek nagyobb számban ismétlődnek egy-egy szabályos tartományban, akkor ezeket rendre elegendő egy-egy példányban tárolni, amennyiben az eredeti előfordulási helyeik egy indexfüggvénybe foglalhatók.)