

mikro *μlógia 19*

Szlávi Péter

Zsakó László

*Módszeres programozás:
Programozási tételek*

Módszeres programozás: Programozási tételek

A kötet bevezető fejezetében -a *μlógia 18* kötetében már taglalt- programozási alapfogalmakat foglaljuk össze, és későbbi céljainknak megfelelően formalizáljuk.

Az I. fejezetben kaptak helyet a legelemibb tételfajták, nevezetesen azok, amelyek egy sorozathoz rendelnek egy értéket: a *sorozatszámítás*, az *eldöntés*, a *kiválasztás*, a *keresés* lineáris változata, a *megszámolás* s végül a *maximumkiválasztás*.

Ezt követően rátérünk azokra a "bonyolultabbak" tételekre, amelyeket az jellemez, hogy mind a bemeneten, mind a kimeneten legalább egy sorozat szerepel. Kiindulunk a *másolás* legáltalánosabb tételéből, amelyre az Adatfeldolgozás című mikrológia kötet is épül. A fejezet további öt tétele: *kiválogatás*, *szétválogatás*, *metszet*, *unió* és *egyesítés*.

A harmadik és a negyedik fejezet egy-egy gyakori, "önálló" témát dolgoz föl, úgy- mint *rendezés* és *keresés*. A rendezés témakör csonka lenne, ha összehasonlító hatékonysági jellemzőket nem mellékelnénk. Itt megadjuk az egyes nevezetes (10) algoritmus minimális és maximális lépésszámát (hasonlítások, illetve mozgások száma), valamint az adatok helyfoglalását. Egyes rendezések algoritmusát más mikrológia kötetekben lehet megtalálni. (*μlógia 4* - Quicksort, *μlógia 27* - rendezés bináris fával.)

Az utolsó fejezetben jutunk el a csúcspontra: megvizsgáljuk, hogyan lehet tételeket egymásra építeni. Központi szereplő -érthető okok miatt- a *kiválogatás* tétel lesz.

Szlávi Péter — Zsakó László

Módszeres programozás:

Programozási tételek

Tartalomjegyzék

μlógia 19

ELTE Informatikai Kar

6., bővített kiadás

Készült az **NJSZT** gondozásában
200 példányban
Felelős kiadó: dr. Kozma László
Sorozatszerkesztő: Szlávi Péter - Zsakó László
© Szlávi Péter - Zsakó László, 2004.

Bevezetés	5
I. Elemi programozási tételek	12
1. Sorozatszámítás	12
2. Eldöntés	15
3. Kiválasztás	17
4. (Lineáris) keresés	19
5. Megszámolás	20
6. Maximumkiválasztás	22
II. Összetett programozási tételek	24
1. Másolás	24
2. Kiválogatás	25
3. Szétválogatás	27
4. Metszet	30
5. Egyesítés (unió)	32
6. Összefuttatás (rendezettek uniója)	33
III. Rendezések	36
1. Egyszerű cserés rendezés	36
2. Minimumkiválasztásos rendezés	37
3. Buborékos rendezés	38
4. Javított buborékos rendezés	39
5. Beillesztéses rendezés	39
6. Javított beillesztéses rendezés	40
7. Szétosztó rendezés	41
8. Számlálva szétosztó rendezés	42
9. Számláló rendezés	43
10. Rendezés Shell módszerrel	44
IV. Keresések	45
1. Keresés rendezett sorozatban	45
2. Visszalépéses keresés (backtrack)	47
V. Programozási tételek egymásra építése	57

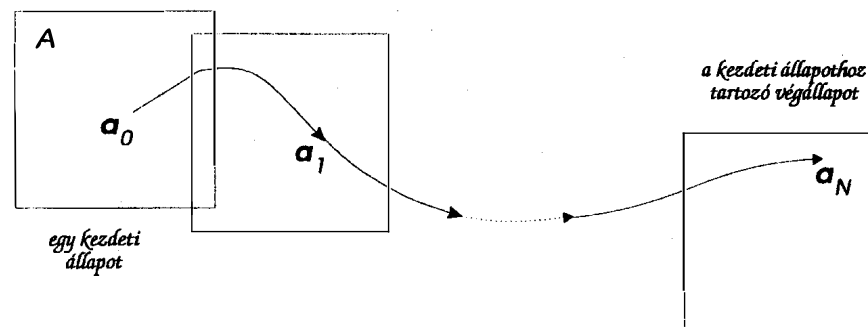
1. Másolással összeépítés	57
2. Megszámolással összeépítés	58
3. Maximumkiválasztással összeépítés	59
4. Kiválogatással összeépítés	61
Irodalomjegyzék	64

Bevezetés

Programozási feladatok megoldásakor a program felülről lefelé való kifejtésekor háromféle programszerkezetet használunk: utasítások egymásutáni végrehajtását, feltételtől függő elágazást, valamint utasítások többszöri megismétlését, ciklust. Ezek választása a megoldandó feladat specifikációjától függ.

A programozási bevezetővel foglalkozó füzetben áttekintettük a specifikáció témakörét, a specifikáció és a programszerkezet kapcsolatát. E kötetben is alapvető szerepet játszanak ezek az ismeretek, így emlékeztetőül röviden áttekintjük őket, illetve a szükségleteknek megfelelően formalizáljuk.

A feladatot mindig valamilyen adatok együttesén fogalmazzuk meg (*ezt ismerjük, erre vagyunk kíváncsiak*), ezen adatok értékhalmozait direkt szorzatát nevezzük állapot-térnek¹. E tér egy pontját -ami a programfutás pillanatnyi rögzítése- nevezzük állapotnak (innen származik a tér neve), koordinátáit pedig változóknak. (Egyes állapotter-koordinátákról kiderülhet a feladatmegoldás során, hogy nem elemi értékkel rendelkeznek.) A változók tehát tulajdonképpen az állapottér projekciói (vetítései).



a program futása: egyes "fázisai"

Példa:

Állapottér: $A = B \times C \times \dots \times H$
 Egy állapota: $a = (b, c, \dots, h)$, ahol $b \in B, c \in C, \dots, h \in H$.
 Egy változója: $\beta: A \rightarrow B$ projekció

¹ Az állapottér a feladatmegoldás során a kezdeti elképzeléshez képest általában bővül; többnyire minden finomítási lépésben.

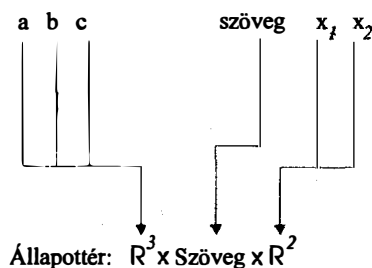
Feladaton mindig egy relációt értünk, amely része az $A \times A$ állapotpárok halmazának ($F \subseteq A \times A$). A fenti ábrán pl. az $(a_0, a_N) \in F$, azaz az F -t a lehetséges kezdőállapotok és a hozzátartozó végállapotok párai alkotják. ($a_0 = (b_0, c_0, \dots, h_0)$, $a_1 = (b_1, c_1, \dots, h_1)$, ... $a_N = (b_N, c_N, \dots, h_N)$) Tehát elsődleges dolog minden feladatnál megállapítani az állapotteret, s az egyes koordinátáinak lehetséges kapcsolatait. Ezután meg kell határozni az állapotternek azt a részét, amelyre a feladat megoldását meg kell kapnunk. Ezt írhatjuk le az **előfeltétellel**; az elérendő eredmény, azaz a cél meghatározására szolgál az **utófeltétel**. (Pusztán technikai okok miatt történik e két A -beli részhalmaz megadása egy-egy feltétellel, azaz matematikai formalizmust fölhasználva egy-egy logikai formulával.)

A fogalmak formális szemléltetésére használjuk ugyanazokat a példákat, amelyeket a bevezető kötetben áttekintettünk!

1. feladat:

Adjuk meg egy legfeljebb másodfokú egyenlet megoldását! Az egyenlet $ax^2 + bx + c = 0$ formában adott. A megoldás(ok) mellett szövegesen is adjuk meg a megoldás milyenségét (két különböző valós megoldás, két egybeeső valós megoldás, nincs valós megoldás, elsőfokú egyenlet egy megoldása)!

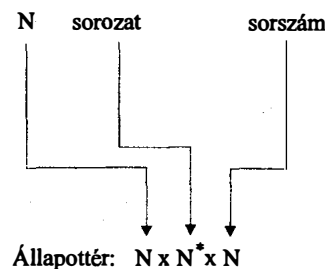
Bemenő paraméterek Kimenő paraméterek



2. feladat (1. változat):

Adjuk meg N ember közül a legmagasabb sorszámát! A magasságokat centiméterben adjuk meg.

Bemenő paraméterek Kimenő paraméter

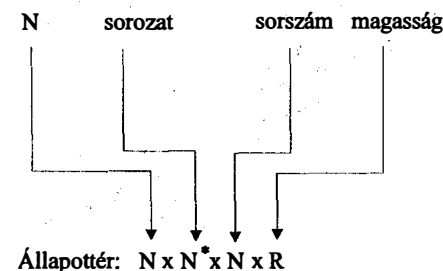


$$\text{ahol } N^* = \bigcup_{k=0}^{\infty} N^k$$

3. feladat (1. változat):

Adjuk meg N ember közül a második legmagasabb sorszámát és magasságát! A magasságokat centiméterben adjuk meg, s az eredményt méterben várjuk.

Bemenő paraméterek Kimenő paraméterek



A feladat pontos megadásához ismernünk kell a bemenő és a kimenő adatok leírását. Azaz a specifikációnak tartalmaznia kell az állapottér egyes komponenseit. Hogy hivatkozni lehessen ezekre: nevezzük el őket! Ezek lesznek egyben a program *bemenő*, ill. *kimenő változói*.

1. feladat:

Bemenet: $a, b, c \in \mathbb{R}$ - együtthatók.

Kimenet: $sz \in \text{Szöveg}$ - a megoldás esetére utaló szöveg

$x_1, x_2 \in \mathbb{R}$ - az egyenlet megoldásai.

2. feladat:

Bemenet: $N \in \mathbb{N}$ - az emberek száma,

$A \in \mathbb{N}^N$ - a magasságukat (cm) tartalmazó tömb.

Kimenet: $MAX \in \mathbb{N}$ - a legmagasabb ember sorszáma.

3. feladat:

Bemenet: $N \in \mathbb{N}$ - az emberek száma,

$A \in \mathbb{N}^N$ - a magasságukat (cm) tartalmazó tömb.

Kimenet: $MAX2 \in \mathbb{N}$ - a második legmagasabb ember sorszáma,
 $MAG2 \in \mathbb{R}$ - a második legmagasabb ember magassága (m).

Most a fenti absztrakt fogalmakat kissé gyakorlatiasabb szemszögből tekintve újra fogalmazzuk. A specifikációnak tartalmaznia kell a **feladatban használt fogalmak definícióját**, valamint az **eredmény kiszámítási szabályát**. A bemenő, illetve a kimenő adatokra kirótt feltételeket neveztük **előfeltételnek**, illetve **utófeltételnek**. Mint az alábbiakból kiderül: szükség esetén bővítjük az állapoteret új (munka-) változókkal.

1. feladat:

Állapottér kibővítve: $\mathbb{R}^3 \times \text{Szöveg} \times \mathbb{R}^2 \times \mathbb{R}$.

Új változó: $d \in \mathbb{R}$ - diszkrimináns.

Előfeltétel: $a \neq 0$ vagy $b \neq 0$.

Utófeltétel: $d = b^2 - 4 \cdot a \cdot c$ és

($a = 0 \Rightarrow x_1 = -c/b$, $sz = \text{"elsőfokú az egyenlet, egy megoldása van"}$) és

($a \neq 0$ és $d < 0 \Rightarrow sz = \text{"nincs valós megoldás"}$) és

($a \neq 0$ és $d = 0 \Rightarrow x_1 = -b/(2 \cdot a)$, $sz = \text{"egy kétszeres valós megoldás van"}$) és

($a \neq 0$ és $d > 0 \Rightarrow x_1 = (-b + \sqrt{b^2 - 4 \cdot a \cdot c})/(2 \cdot a)$, $x_2 = (-b - \sqrt{b^2 - 4 \cdot a \cdot c})/(2 \cdot a)$,
 $sz = \text{"két különböző valós megoldás van"}$).

2. feladat:

Előfeltétel: -.

Utófeltétel: $1 \leq MAX \leq N$ és $\forall i (1 \leq i \leq N): A(MAX) \geq A(i)$.

3. feladat:

Állapottér kibővítve: $\mathbb{N} \times \mathbb{N}^* \times \mathbb{N} \times \mathbb{R} \times \mathbb{N}$.

Új változó: $MAX \in \mathbb{N}$ - a legmagasabb sorszáma.

Előfeltétel: $N \geq 2$ és $\exists i, j (1 \leq i, j \leq N): A(i) \neq A(j)$.

Utófeltétel: $1 \leq MAX \leq N$ és $\forall i (1 \leq i \leq N): A(MAX) \geq A(i)$ és

$1 \leq MAX2 \leq N$ és $\forall i (1 \leq i \leq N): (A(MAX2) \geq A(i) \text{ vagy } A(MAX2) < A(MAX))$ és

$MAG = A(MAX2)/100$.

A feladatot megoldó algoritmus, illetve program természetesen lehet a kitűzött feladatnál **általánosabb**. Ez azt jelenti, hogy a program **utófeltétele lehet erősebb**, azaz a feladat által meghatározott helyes eredmények halmazánál szűkebbet adó, **előfeltétele pedig gyengébb**, azaz a feladat által meghatározott bemenő adatoknál bővebb körre is működhet.

A most következő rövid okfejtésből látszik majd, hogy a specifikáció megfelelően precíz megadása komoly segítséget jelenthet a későbbi algoritmizálásnál. Induljunk ki abból, hogy a feladatot megoldó program valamilyen leképezést valósít meg a bemenő és a kimenő adatok között. Ezt a leképezést, amely sokszor egyértelmű, hívják **programfüggvénynek**. A programfüggvény definiálására használjuk a függvények körében

szokványosnak mondható formális eszközöket, úgymint: az egymásba ágyazás, vagy függvénykompozíció, az alternatíva és a rekurzió. Egy egyszerű példán is könnyen belátható, hogy a függvénytüveletek és az algoritmikus szerkezetek között a következő kapcsolat áll fenn:

függvényművelet		algoritmikus szerkezet	
kompozíció	$h(x) := (f \circ g)(x)$	$hb := g(x)$ $h := f(hb)$	szekvencia
alternatíva	$h(x) := \begin{cases} f(x), & \text{ha } T(x) \\ g(x), & \text{különben} \end{cases}$	Ha $T(x)$ akkor $h := f(x)$ különben $h := g(x)$	elágazás
rekurzió	$h(x) := \begin{cases} f(x), & \text{ha } T(x) \\ (g \circ h \circ i)(x), & \text{különben} \end{cases}$	$k := 0$ Ciklus amíg nem $T(x)$ $x := i(x)$; $k := k + 1$ Ciklus vége $x := x$; $hx := f(x)$ Ciklus $l = 1$ -től k -ig $hx := g(hx)$ Ciklus vége $h := hx$	ciklus

A **strukturális feldolgozás** -már korábban megismert²- elve további támpontokat nyújt az algoritmus formálásához. Emlékezzünk, hogy az adatok szerkezete és az őket feldolgozó programszerkezet között is -az előbbihez hasonló könnyedséggel belátható- viszony található:

adatszerkezet		programszerkezet
skalár	„elemi adat”	értékkadás (függvény), eljárás
direktszorzat	rekord	mezők szerinti szekvencia
iteráció	sorozatféle=sokaság	„elemfeldolgozó magú” ciklus
unió	alternatív rekord	feltétel szerinti elágazás

Ezek a felismerések, elvek sem adnak azonban segítséget abban, hogy teljes precíziásban „generáljuk” a program algoritmusát. Hogy ebben az értelemben előbbre juthassunk, fel kell ismernünk a programozási feladatokban rejlő lényeges jellemzőt: a **feladatok nagy feladatosztályokba sorolhatók a feladat jellege szerint, s e feladatosztályokhoz tudunk készíteni a teljes feladatosztályt megoldó algoritmosztályt**. Ezeket a típusfeladatokat **programozási tételeknek** fogjuk nevezni, ugyanis bebizonyítható (s ezt meg is tesszük), hogy a megoldásai a feladat garantáltan helyes megoldásai.

² Természetesen "algoritmikus rekurzióval" még triviálisabb megoldást lehet találni. (Speciális esetekben lényegesen egyszerűbb az iteratív változat.)

E programozási tételek ismeretében már a legtöbb feladat megoldása során nincs más dolgunk, mint felismerni a megfelelő programozási tételt, megfeleltetni a konkrét adatait az általános feladatosztálynak, majd ezeket az általános megoldó algoritmusban a konkrétumokkal helyettesíteni. Ezzel a módszerrel a feladatnak mindig egy -elvileg- helyes megoldást kapjuk³, azonban ez nem mindig a leghatékonyabb, legegyszerűbb, legügye-sebb megoldás lesz. A helyes megoldásból a különféle szempontok szerinti hatékony megoldáshoz vezető úttal a sorozat egy másik kötete⁴ foglalkozik.

Nilvánvalónak látszik, hogy a programfüggvény három lehetséges esete közül a ciklussal (illetve rekúzióval) megoldandó feladatok lesznek a legnehezebbek. A programo-zási tételek éppen ezért az ilyen feladattípusokhoz kapcsolódnak.

Ezek a feladatok mindig olyanok lesznek, hogy egy (vagy több) adatsokasághoz kell rendelnünk valamilyen eredményt. E sokaságot az egyszerűség kedvéért mindig valami-lyen sorozatként fogjuk fel. A „sorozatság” az elemek feldolgozási sorrendjét határozza meg.⁵ A legtöbb esetben elég olyan sorozatokkal foglalkozni, amelyek elemei egymás után -egyesével- feldolgozhatók. Ez a bemenő sorozatra olyan művelet létét feltételezi, amely előlről, egyenként képes adni a sorozat elemeit, az eredmény sorozatba pedig az addig belekerült elemek mögé képes tenni egy új elemet. További egyszerűsítést veze-tünk be a megfogalmazásba: a sorozatokat mindig egy tömbbel⁶ ábrázoljuk.

A vizsgált feladatosztályokat 4 típusra osztjuk a bemenet és a kimenet alapján:

- egy sorozathoz egy értéket rendelő feladatok,
- egy sorozathoz egy sorozatot rendelő feladatok,
- egy sorozathoz több sorozatot rendelő feladatok,
- több sorozathoz egy sorozatot rendelő feladatok.

Mindegyik programozási tétel tárgyalását néhány konkrét példával kezdjük. E példák alapján határozzuk meg a feladattípus általános jellemzőit. Ezután definiáljuk az általá-nos megoldásban használt adatokat mind szöveges leírással, mind pedig formális (mate-matikai) eszközökkel leírva. A formális leírásokat a szövegből keretezéssel kiemeltük, a tételek ezek kihagyásával is megérthetők.

Következő lépésként áttérünk a megoldás kialakítására, majd folytatjuk az általános megoldás algoritmusával. Ha a tételnek többféle variációját tárgyaljuk, akkor itt ezek következnek, s befejezésként megoldunk néhányat a fejezet elején közölt példákból.

JELÖLÉSEK

A könyvben használunk néhány jelölést, ezért a továbblépés előtt tekintsük át ezeket:

\mathbf{N}_0	- 0, 1, 2, ... (nem negatív egész számok)
H	- tetszőleges halmaz (halmazműveletekkel)
H^* <i>H-beli véges sorozatok hal-maza</i>	- $H^* = \bigcup_{k=0}^{\infty} H^k$ (i. elemére h_i -vel hivatkozunk)
H^N	- <i>H-beli N elemű sorozatok halmaza</i>
L	- <i>logikai értékek halmaza</i> : {igaz, hamis}
\bar{H} <i>rendezett halmaz</i>	- $\forall h_1, h_2 \in H: h_1 \leq h_2$ vagy $h_1 \geq h_2$

Megjegyzés:

ha egy H halmazról *rendezettséget* feltételezünk, akkor azt így írjuk le: *Rendezett(H)*.

S <i>rendezett sorozat</i>	- $\forall i (1 \leq i < N): s_i \leq s_{i+1}$ ⁷
$S \in H^*$ <i>halmazfelsorolás</i>	- $\forall i, j (1 \leq i, j \leq N): i \neq j \Rightarrow s_i \neq s_j$
$X, Y \in H^*: X \subseteq Y$ <i>X részsorozata Y-nak</i>	- $X = (x_1, \dots, x_M) = (y_1, \dots, y_M) = Y$ és $i_1 < \dots < i_M$ (természetesen: $M \leq N$)
$X, Y \in H^*: X \& Y$ <i>két sorozat egymásután írása</i>	- $X \& Y = (x_1, \dots, x_N, y_1, \dots, y_M)$
R halmaz: $S = \text{Permutáció}(R)$ <i>S R permutációja</i>	- $R = (r_1)$ és $S = (s_1)$ vagy $S = (s_1) \& S'$ és $s_1 \in R$ és $S' = \text{Permutáció}(R \setminus s_1)$
$X, Y, Z \in H^*: X = Y \cup Z$ <i>Y és Z az X diszjunkt felbon-tása</i>	- $X = Y \& Z$ és $Y \cap Z = \emptyset$ ⁸

³ Természetesen ez nem véd a hibás gondolkodástól, illetve a kódolás során elkövetett hibák ellen.

⁴ L. *ulógia 6* - Módszeres programozás: Hatékonyság.

⁵ Azaz feltételezzük, hogy létezik elemei között egy egyértelmű *rákövetkezési reláció* és ezen nyugvó elemeléréshez van valamilyen definiált *szelekciós*, illetve *konstrukciós függvény*.

⁶ Adott, *rögzített elemszámú sorozat*, *indexelés* művelettel.

⁷ Sokszor kényszerülünk arra, hogy az S sorozat i elemét S_i helyett $S(i)$ -vel jelöljük. Ez ne okozzon zavart.

⁸ azaz az X minden eleme pontosan egyszer szerepel az Y vagy a Z sorozatban.

I. Elemi programozási tételek

Feladataink egy jelentős csoportjában egyetlen bemenő sorozat alapján kell meghatározunk egy értéket eredményként.

Bemenet : $N \in \mathbb{N}_0, X \in H^N, F: H^* \rightarrow G$ ← az állapottér bemeneti komponensei
 Kimenet : $S \in G$ ← az állapottér bemeneti komponensei
 ef : - ← Nincs előfeltétel (ef),
 uf : $S = F(X_1, \dots, X_N)$ ← Utófeltétel (uf), amilyen tulajdonsággal megálláskor az S rendelkezni fog. (Most hogy éppen az adott értékű lesz.)

Az elő-, illetve az utófeltételben pontosan meg kellene fogalmaznunk azt is, hogy a bemenő adatok, paraméterek értéke a megoldás során változatlan marad, ettől azonban az egyszerűbb felírás miatt -általában- eltekintünk. A pontos használat miatt -mintaként- ezen a helyen így is megadjuk. A *apostrof* jelöli a paraméter, bemenő változó régi értékét.

Bemenet : $N \in \mathbb{N}_0, X \in H^N, F: H^* \rightarrow G$
 Kimenet : $S \in G$
 ef : -
 uf : $S = F(X_1, \dots, X_N)$ és $N = N'$ és $X = X'$

Ez a csoport több feladattípust tartalmaz, nézzük végig ezeket!

1. Sorozatszámítás

Kezdjük a legelső témakört néhány feladattal!

F1. Egy osztály N db tanuló osztályzatának ismeretében adjuk meg az osztály átlagát!

F2. Egy M elemű betűsorozat betűit fűzzük össze egyetlen szöveg típusú változóba!

F3. Készítsünk algoritmust, amely egy autóversenyző körönkénti ideje alapján meghatározza a versenyző egy kör megtételéhez szükséges átlagidejét!

F4. A Balaton mentén K db madarász végzett megfigyeléseket. Mindegyik megadta, hogy milyen madarakat látott. Készítsünk algoritmust, amely megadja a megfigyelések alapján a Balatonon előforduló madárfajokat!

F5. Adjuk meg az első N természetes szám szorzatát (N faktoriális)!

Vizsgáljuk meg, mi a közös a fenti öt feladatban! Mindegyikben adatok valamilyen sorozatával kell foglalkoznunk, e sorozathoz kell hozzárendelnünk egyetlen értéket. Ezt az értéket egy, az egész sorozaton értelmezett függvény adja (N szám összege, M betű egymásutánírása, K halmaz uniója, N szám szorzata). Ezt a függvényt azonban felbont-hatjuk értékpárokon kiszámított függvények sorozatára (2 „valami” összegére, egymás-után írására, uniójára, szorzatára).

Az algoritmus:

Változók:

N : Egész [a feldolgozandó sorozat elemei száma]
 X : Tömb(1..N:Elemtípus) [a feldolgozandó sorozat elemei]
 F0: Elemtípus₁ [a művelet nulleleme]
 S : Elemtípus₂ [az eredmény]

Bemenet : $N \in \mathbb{N}_0, X \in H^N, F: H^* \rightarrow G$
 Kimenet : $S \in G$
 ef : $\exists F_0 \in G$ (nullelem) és $\exists f: G \times H \rightarrow G$ és
 $F(X_1, \dots, X_N) = f(F(X_1, \dots, X_{N-1}), X_N)$ és $F() = F_0$
 uf : $S = F(X_1, \dots, X_N)$

Így tehát minden olyan művelet szerepelhet e feladattípusban, amelyre a matematika valamilyen „egységes” jelölést használ: összeg, szorzat, unió, metszet, logikai művele-tek, konkatenáció, Mindegyik művelet visszavezethető egy bináris műveletre, s meg-adható mindegyikhez egy semleges elem (nullelem) is, amelyre egy tetszőleges elemmel és vele elvégzett 2 operandusú művelet az adott elemet adja.

Variációk: $F = \Sigma, \Pi, \cup, \cap, \wedge, \vee, \&$ (konkatenáció).

$F_0 = 0, 1, \{ \}, \text{alaphalmaz, Igaz, Hamis, ""}$.

A megoldás a nullelemre, valamint a 2 operandusú műveletre épül, a matematikában is szokásos módszer, az indukció alapján:

- 0 elemre tudjuk az eredményt: F_0 ,
- ha i-1 elemre tudjuk az eredményt (F^{i-1}), akkor i elemre (F^i) a következőképpen kell kiszámítani: $F^i = f(F^{i-1}, X_i)$.

Az F-re vonatkozó alábbi rekurzív definícióból:

$$F_i := F(X_1, \dots, X_i) := \begin{cases} F_0 & , \text{ ha } i = 0 \\ f(F^{i-1}, X_i) & , \text{ különben} \end{cases}$$

–a korábbi alapján, a specialításokat figyelembe véve– így kapható az algoritmus első változata:

a bevezetésbeli jelölésekkel	a specialítások figyelembe vétele
<pre> k:=0 Ciklus amíg nem T(x) x:=i(x): k:=k+1 Ciklusvége x*:=x: hx:=f(x*) Ciklus l=1-től k-ig hx:=g(hx) Ciklus vége h:=hx </pre>	<pre> [T(x)≡i=0] k:=N: x:=() [i:(X₁, .. X_i) → (X₁, X_{i-1})] x*:=(): hx:=F [h()=F] hx:=f(hx,X) F:=hx </pre>

A „főlöleges” dolgokat kihagyva mindössze ennyi marad:

```

hx:=F0
Ciklus l=1-től N-ig
  hx:=f(hx,X)
Ciklus vége
F:=hx

```

Végezetül az algoritmikus nyelvünkben szokásos jelölésekkel a feladatot megoldó algoritmus:

```

Sorozatszámítás(N,X,S):
  S:=F0
  Ciklus I=1-től N-ig
    S:=f(S,X(I))
  Ciklus vége
Eljárás vége.

```

Alkalmazzuk az általános megoldást a fejezet elején kitűzött egyes feladatokra!

F1. N szám összege

```

elemek száma: N
osztályzatok: X N elemű
vektor
F, f, F0 : S, +, 0

```

```

Összegzés(N,X,S):
  S:=0
  Ciklus I=1-től N-ig
    S:=S+X(I)
  Ciklus vége
Eljárás vége.

```

F2. M betű egymásutánírása

```

elemek száma : M
betűk       : X M elemű
vektor
F, f, F0 : &, +, "",

```

```

Egymásután(M,X,SZ):
  SZ:="" [üres szöveg]
  Ciklus I=1-től M-ig
    SZ:=SZ+X(I)
  Ciklus vége
Eljárás vége.

```

F4. K halmaz uniója

```

elemek száma : K
megfigyelések: X N db elem
F, f, F0 : ∪, ∪, {}

```

```

Unió(K,X,H):
  H:={} [üres halmaz]
  Ciklus I=1-től K-ig
    H:=H∪X(I)
  Ciklus vége
Eljárás vége.

```

2. Eldöntés

Az előző programozási tétel két speciális esetében az ott közölt megoldás sok felesleges lépést tartalmazhat. Ez a tétel annak a hiányosságait pótolja e speciális esetekben. Az alábbi példákban keressük meg a közös vonást (ami speciális esetét jelenti az előzőnek)!

F6. Döntsük el egy számról, hogy prímszám-e!

F7. Döntsük el egy szóról a hónapnevek sorozata alapján, hogy egy hónap neve-e!

F8. Döntsük el egy tanuló év végi osztályzatai alapján, hogy kitűnő tanuló-e!

F9. Júniusban minden nap délben megmértük, hogy a Balaton Siófoknál hány fokal. Döntsük el a mérések alapján, hogy a víz hőfoka folyamatosan emelkedett-e!

E feladatok közös jellemzője, hogy a bemenetként megadott sorozathoz egy logikai értéket kell rendelni: a feltett kérdésre **igennel** vagy **nem**-mel kell válaszolni. A vizsgált sorozat elemei tetszőlegesen lehetnek, egyetlen jellemzőt kell feltételeznünk róluk: egy tetszőleges elemről el lehet dönteni, hogy rendelkezik-e egy bizonyos tulajdonsággal.

A feladatok így két csoportba sorolhatók, az egyikben azt kell eldönteni, hogy egy sorozatban létezik-e adott tulajdonságú elem, a másikban pedig azt, hogy mindegyik elem rendelkezik-e ezzel a tulajdonsággal. Vizsgáljuk először az elsőfajtaúakat!

```

Bemenet : N ∈ N0, X ∈ HN, T: H → L
Kimenet : VAN ∈ L
ef       : -
uf       : VAN ≡ (∃i (1 ≤ i ≤ N): T(Xi))

```

1. algoritmus:

Függvény:
T: Elemtípus → Logikai

Változók:

N : **Egész** [a feldolgozandó sorozat elemei száma]
 X : **Tömb**(1..N:Elemtípus) [a feldolgozandó sorozat]
 VAN: **Logikai** [az eredmény]

A feladat típus megoldására az előző programozási tétel is alkalmas a következő megfeleltetéssel: $F = \bigvee, f = \bigvee, F_0 = \text{hamis}$.

Eldöntés(N, X, S) :

S:=hamis
Ciklus I=1-től N-ig
 S:=S \vee X(I)
Ciklus vége
Eljárás vége.

Észrevehetünk azonban egy fontos tulajdonságot. Ha a megoldásban az S változó értéke egyszer **igazra** változik, akkor a megoldás végéig biztosan az is marad. Tehát az ezutánani műveletek elvégzése teljesen felesleges. Ezt a tulajdonságot kihasználva készíthetjük el az igazi megoldást.

Ebben az esetben egy olyan ciklus a megoldás, amely vagy akkor áll le, ha találtunk egy, a keresett tulajdonsággal rendelkező elemet, vagy pedig akkor, ha ilyen elem a sorozatban már nem létezhet, azaz elfogytak a megvizsgálандó elemek.

Eldöntés(N, X, VAN) :

I:=1
Ciklus amíg I≤N és nem T(X(I))
 I:=I+1
Ciklus vége
 VAN:=(I≤N)
Eljárás vége.

Fordítsuk most figyelmünket a másik csoportra! Azt, hogy mindegyik elem rendelkezik egy adott tulajdonsággal, átfogalmazhatjuk arra, hogy nem létezik az adott tulajdonsággal nem rendelkező elem. Ezek alapján a fenti megoldásban 2 helyen tagadást alkalmazva megkapjuk ennek a csoportnak a megoldástípusát is. (Ez a sorozatszámítás az $F = \bigwedge, f = \bigwedge, F_0 = \text{igaz}$ értékekkel).

Eldöntés(N, X, MIND) :

I:=1
Ciklus amíg I≤N és T(X(I))
 I:=I+1
Ciklus vége
 MIND:=(I>N)
Eljárás vége.

Az eldöntés tipikusan olyan feladat, amelynél kódolási problémák merülhetnek fel. Ilyenekkel foglalkozunk az algoritmuskódolásával kapcsolatos fejezetben, itt csupán utalunk a problémára és a lehetséges megoldásaira.

A programozási nyelvek egy jelentős részében a logikai kifejezések kiértékelésekor az **és**, illetve a **vagy** műveletek mindkét operandusát kiértékelik futáskor, még akkor is,

ha az egyik operandus értéke alapján a végeredmény egyértelműen eldönthető lenne. E programozási tételnél így az $I \leq N$ feltétel nem teljesülése esetén is meg kell vizsgálni a $T(X(I))$ értékét. Ha a felhasznált tömb pontosan N elemű, akkor ez tömbindexelési hibához vezet. Többféle megoldással foglalkoztunk a kódolási kérdések kapcsán:

- vegyünk fel egy fiktív (N+1.) elemet a tömb végére,
- a ciklus egy lépéssel hamarabb álljon le, s vizsgáljuk a leállás okát,
- válasszuk szét a két részfeltétel kiértékelését, s a $T(X(I))$ -t csak akkor értékeljük ki, amikor szükséges.

Néhány feladat megoldása következik.

F6. Létezik-e a számnak 1-től és önmagától különböző osztója?

elemek száma : N-2
 osztók : 2, 3, ..., N-1
 T(e)=T'(e, N) : e|N
 (T'(e, N) : a valódi függést
 kifejező leképezés)

Eldöntés(N, PRIM) :
 I:=2
Ciklus amíg I≤N-1 és
 nem I|N
 I:=I+1
Ciklus vége
 PRIM:=(I>N-1)
Eljárás vége.

F9. A hőmérsékletek sorozata monoton növekedő-e?

elemek száma : N-1
 hőmérsékletek: X N db elem
 T(X(e))=T'(e) : X(e)≤X(e+1)

Eldöntés(N, X, MONOTON) :
 I:=1
Ciklus amíg I≤N-1 és
 X(I)≤X(I+1)
 I:=I+1
Ciklus vége
 MONOTON:=(I>N-1)
Eljárás vége.

3. Kiválasztás

Ha kicsit belegondolunk, az előző programozási tétel megoldása a tőle vártól több eredményt is adhat. Ennél a tételnél ezt a *többet* vizsgáljuk.

F10. Ismerjük egy hónap nevét, a hónapnevek sorozata alapján mondjuk meg a sorszámat!

F11. Adjuk meg egy természetes szám legkisebb, 1-től különböző osztóját!

F12. A naptárban található névnapok alapján adjuk meg legjobb barátunk (barátunk) névnapját! (Itt nem a „legjobb” megfogalmazása okozza az algoritmikai problémát.)

Keressük itt is a közös jellemzőket! A feladattípus első ránézésre nagyon hasonlít az előzőre, csupán itt nem egy eldöntendő kérdésre kell válaszolni, hanem meg kell adni a sorozat egyik, adott tulajdonsággal rendelkező elemét. Kicsit részletesebb, pontosabb vizsgálódás után még az is kiderül, hogy ha e feladatokra az eldöntendő kérdést tennénk fel, akkor biztosan igen választ kapnánk.

Azt kell még eldöntenünk, hogy az elemet hogyan adhatjuk meg. Erre két lehetőségünk van: vagy a sorszámát, vagy pedig az értékét adjuk meg. Felhívjuk a figyelmet arra, hogy az előbbi változatból az utóbbi megoldását nagyon könnyen megkaphatjuk, fordítva viszont korántsem ez a helyzet. Emiatt mi az első változatot részesítjük előnyben, ez az általánosabb eset.

Bemenet : $N \in \mathbb{N}_0, X \in H^N, T: H \rightarrow L$
 Kimenet : $SORSZ \in \mathbb{N}$
 ef : $\exists i (1 \leq i \leq N) : T(X_i)$ [és természetesen: $N \geq 1$]
 uf : $1 \leq SORSZ \leq N$ és $T(X_{SORSZ})$

Az algoritmus:

Függvény:

T : Elemtípus \rightarrow Logikai

Változók:

N : **Egész** [a feldolgozandó sorozat elemei száma]
 X : **Tömb**($1..N$:Elemtípus) [a feldolgozandó sorozat elemei]
 $SORSZ$: **Egész** [az eredmény]

A megoldás sokban fog hasonlítani az előző feladattípus megoldásához, annak az esetnek a vizsgálatát kell kihagyni belőle, amely a keresett tulajdonságú elem nem létezése esetén állította le a megoldás keresését.

Kiválasztás ($N, X, SORSZ$):

$I := 1$
Ciklus amíg nem $T(X(I))$
 $I := I + 1$
Ciklus vége
 $SORSZ := I$

Eljárás vége.

A megoldásról könnyen megállapíthatjuk, hogy ha több, a kívánt tulajdonsággal rendelkező elem is előfordul a sorozatban, akkor azok közül az elsőt (helyesebben annak sorszámát) adja.

Ebből, mint azt egy korábbi fejezetben a specifikáció kapcsán már taglaltuk, az következik, hogy a *program* utófeltétele lehet szigorúbb mint a feladatbeli utófeltétel. Itt ez a következőképpen néz ki:

uf: $1 \leq SORSZ \leq N$ és $T(X_{SORSZ})$ és $\forall i (1 \leq i < SORSZ) \neg T(X_i)$

Könnyen átalakíthatjuk olyanra is, amely ilyenkor az utolsót adja közülük. Ekkor csupán annyi a teendő, hogy a sorozat elemeit hátulról visszafelé dolgozzuk fel.

Nézzük meg a bevezető feladatok egyikének megoldását!

F12. A legjobb barát (barátnő) névnapja.

elemek száma: N
 névnapok : X N db elem
Rekord(név, névnap)
 $T(e)$: $e.név = barát$

Kiválasztás ($N, X, BARÁT, NAP$):
 $I := 1$
Ciklus amíg $X(I).név \neq BARÁT$
 $I := I + 1$
Ciklus vége
 $NAP := X(I).névnap$
Eljárás vége.

4. (Lineáris) keresés

Foglaljuk össze az előző két programozási tétel feladatát: az új feladattípusban a feltett kérdés az előző kettő mindegyikét tartalmazza!

F13. Ismerjük egy üzlet egy havi forgalmát: minden napra megadjuk, hogy mennyi volt a bevétel és mennyi a kiadás. Adjunk meg egy olyan napot -ha van-, amelyik nem volt nyereséges!

F14. A Budapest-Nagykanizsa vasúti menetrend alapján két adott állomáshoz adjunk meg egy olyan vonatot, amellyel el lehet jutni átszállás nélkül az egyikről a másikra!

F15. Egy tetszőleges (nem 1) természetes számnak adjuk meg egy osztóját, ami nem az 1 és nem is önmaga.

Tehát a közös jellemző, hogy egy adott tulajdonságú elemet kell megadnunk, ha egyáltalán van ilyen. Ha nincs, akkor a válasznak ezt a tényt kell tartalmaznia.

Bemenet : $N \in \mathbb{N}_0, X \in H^N$
 $T: H \rightarrow L$
 Kimenet : $VAN \in L, SORSZ \in \mathbb{N}$
 ef : -
 uf : $VAN \equiv (\exists i (1 \leq i \leq N) : T(X_i))$ és
 $VAN \Rightarrow 1 \leq SORSZ \leq N$ és $T(X_{SORSZ})$

Az algoritmus:

Függvény:

T : Elemtípus \rightarrow Logikai

Változók:

```
N: Egész [a feldolgozandó sorozat elemei száma]
X: Tömb(1..N:Elemtípus) [a feldolgozandó sorozat elemei]
VAN: Logikai [az eredmény - van-e megfelelő elem]
SORSZ: Egész [az eredmény - a megfelelő elem sorszáma]
```

Vegyük az *eldöntés* algoritmusát, s egészítsük ki a *kiválasztás*nak megfelelő eredménnyel!

Keresés (N, X, VAN, SORSZ) :

```

I:=1
Ciklus amíg I≤N és nem T(X(I))
    I:=I+1
Ciklus vége
VAN:=(I≤N)
Ha VAN akkor SORSZ:=I
ljárás vége.

```

Az egyik kitűzött feladat megoldása a következő lehet:

F13. Nem nyereséges nap megadása.

```

elemek száma: N
bevételek      : B N db elem
kiadások       : K N db elem
T(k-b)         : k-b ≥ 0

```

```

Keresés(N,X,VAN,NAP):
  I:=1
  Ciklus amíg I≤N és
                                K(I)−B(I)<0
    I:=I+1
  Ciklus vége
  NAP:=X(I).névnap
Eljárás vége.

```

5. Megszámolás

Az előző feladatokban előfordulhatott, hogy több elem is rendelkezik a vizsgált tulajdonsággal. Ekkor érdekes lehet annak megvizsgálása, hogy hány ilyen elem van.

F16. Családok létszámának, illetve jövedelmének alapján állapítsuk meg, hogy hány család él a létminimum alatt!

F17. Egy futóverseny végeredménye határozzuk meg, hogy a versenyzők hány százaléka teljesítette az olimpiai induláshoz szükséges szintet!

F18. Adjuk meg egy szöveg magánhangzóinak számát!

A közös jellemző itt tehát a számlálás. Vegyük észre, hogy a feladatot sorozatszámításként is felfoghatjuk, amelyben 1-eseket kell összeadnunk, de bizonyos feltételtől függően.

Bemenet : $N \in \mathbf{N}_0, X \in H^N$

$$T: H \rightarrow L, \quad \chi: H \rightarrow \{0, 1\}, \quad \chi(x) = \begin{cases} 1, & \text{ha } T(x) \\ 0, & \text{egyébként.} \end{cases}$$

$$\begin{aligned} \text{Kimenet} &: \text{DB} \in \mathbf{N}_0 \\ \text{ef} &: - \\ \text{uf} &: \text{DB} = \sum_{i=1}^N \chi(X_i) \end{aligned}$$

z algorithm:

Függvény:

T: Elemtípus \rightarrow Logikai

változók:

```
N : Egész           [a feldolgozandó sorozat elemei száma]
X : Tömb(1..N:Elemtípus) [a feldolgozandó sorozat elemei]
DB: Egész           [az eredmény - a megfelelő elemek száma]
```

A feladat sorozatszámítás (sőt összegzés), tehát egy ciklust kell alkalmazni a megoldáshoz. A ciklus belsejében egy unió típusú (χ függvény értéke) adatot kell feldolgozni, ezt egy elágazással tehetjük meg.

Megszámolás (N, X, DB) :

```
DB:=0
Ciklus I=1-től N-ig
    Ha T(X(I)) akkor DB:=DB+1
Ciklus vége
Végzés vége.
```

Nézzük meg két feladat megoldását!

F16. Létf minimum alattiak száma.

Minden családi létszámhoz megadható az a jövedelem, amely a létminimumhoz kell, a megoldásban ezt használjuk fel.

```

elemek száma: N
létszámok      : L N db elem
jövedelmek     : J N db elem
létminimumok: MIN sorozat
T(1-j)         : j-MIN(1) ≥ 0

```

```

Megszámolás (N, J, L, DB) :
  DB:=0
  Ciklus I=1-től N-ig
    Ha J(I) ≤ MIN(L(I))
      akkor DB:=DB+1
  Ciklus vége
Eljárás vége.

```

F17. Olimpiai indulási szintet hány százalék teljesítette?

A feladat megoldása egy megszámlálás, majd az eredményből és a résztvevők számából százalékot számolunk.

```

elemek száma: N
idők       : ID N db elem
T(id)      : id ≤ SZINT

```

```

Megszámolás (N, ID, SZAZ) :
  DB:=0
  Ciklus I=1-től N-ig
    Ha ID(I) ≤ SZINT
      akkor DB:=DB+1
  Ciklus vége
  SZAZ:=Kerekít(100*DB/N)
Eljárás vége.

```

6. Maximumkiválasztás

A sorozatszámítás egy újabb speciális esetével ismerkedünk meg a következő feladattípusban, először néhány feladaton keresztül.

F19. Egy kórházban megmérték minden beteg lázát, adjuk meg, hogy ki a leglázasabb!

F20. Egy család havi bevételei és kiadásai alapján adjuk meg, hogy melyik hónapban tudtak a legtöbbet megtakarítani!

F21. Egy osztály tanulói nevei alapján adjuk meg a névsorban legelső tanulót!

Közös jellemzőjük e feladatoknak, hogy minden esetben egy sorozat elemei közül kell kiválasztani a legnagyobbat (illetve a legkisebbet). Itt is meggondolandó -mint a *kiválasztásnál*-, hogy elem értéket vagy pedig elem sorszámot várunk eredményként. Az ottani gondolatmenethez hasonlóan most is a sorszám meghatározását választjuk.

```

Bemenet :  $N \in \mathbb{N}_0$ ,  $X \in H^N$ , H rendezett halmaz ( $\exists <, \leq$  reláció)
Kimenet :  $MAX \in \mathbb{N}$ 
ef       :  $N \geq 1$ 
uf       :  $1 \leq MAX \leq N$  és  $\forall i (1 \leq i \leq N): X_{MAX} \geq X_i$ 

```

Az algoritmus:

Változók:

N : **Egész** [a feldolgozandó sorozat elemei száma]
X : **Tömb**(1..N:Elemtípus) [a feldolgozandó sorozat elemei]
MAX: **Egész** [a maximális értékű elem sorszáma]

A sorozatszámításnál használjuk F függvényként a $MAX(A_1, A_2, \dots, A_N)$ függvényt! Ehhez az f függvényt a következőképpen használjuk:

$$f(x, y) := \max(x, y).$$

Legyen ehhez $F_0 := A_1$! A választás megengedhető, annak ellenére, hogy nem feltétlenül lesz az A_1 *neutrális* a *max* műveletre nézve! Ugyanis a korrekt választás a H minimum eleme lenne, ami vagy ismert, vagy nem. Egy tetszőleges A-beli elem *neutrálisnak* akkor fog tűnni, ha a *max* operátor értelmezési tartományát leszűkítjük a H azon rész-

halmazára, amely az adott elemnél nem kisebb értékű elemeket tartalmazza. Ennek is eleme lesz garantáltan a sorozat maximális eleme. Tehát hibát nem követünk el, ha F_0 -nak az A_1 -t tekintjük. Alakítsuk át úgy a sorozatszámítás megoldását, hogy ne értéket, hanem sorszámot kapjunk eredményül! (Annyi észre venni valónk van csak az algoritmizáláskor, hogy most a sorozat a 2. elemmel kezdődik.)

```

Maximumkiválasztás (N, X, MAX) :
  MAX:=1
  Ciklus I=2-től N-ig
    Ha X(MAX) < X(I) akkor MAX:=I
  Ciklus vége
Eljárás vége.

```

Sok esetben túlságosan sok időbe kerül a sorozat egy elemének meghatározása. Ekkor olyan megoldást készíthetünk, amely a maximális értéket határozza meg, vagy pedig a sorszámot is és az értéket is.

```

Maximumkiválasztás (N, X, MAX, MAXERT) :
  MAX:=1: MAXERT:=X(1)
  Ciklus I=2-től N-ig
    Ha MAXERT < X(I) akkor MAX:=I: MAXERT:=X(I)
  Ciklus vége
Eljárás vége.

```

Ha a feladat minimumkiválasztás, akkor pedig nincs más teendőnk, mint az elágazás feltételében szereplő < reláció átírása >-ra. Erre példa az egyik kitűzött feladat megoldása.

F21. A névsorban legelső tanuló.

```

elemek száma: N
tanulók neve: X N db elem

```

```

Minimum (N, X, MIN, MEV) :
  MIN:=1: NEV:=X(1)
  Ciklus I=2-től N-ig
    Ha NEV > X(I)
      akkor MIN:=I: NEV:=X(I)
  Ciklus vége
Eljárás vége.

```

II. Összetett programozási tételek

A soronkövetkező feladatípusokban már a bemeneti és a kimeneti oldalon is egy-egy sorozat jelenik meg, majd a későbbiekben valamelyiket több sorozattal helyettesítjük.

1. Másolás

Ezt a feladatípus is konkrét példákkal kezdjük, mint tettük azt az előző fejezetben.

F22. Egy osztály tanulójának átlageredménye alapján határozzuk meg, hogy bizonyítványukba *jeles*, *jó*, *közepes* vagy *elégséges* kerül-e! Tegyük fel, hogy bukott tanuló nincs!

F23. Egy szöveg minden magánhangzóját cseréljük ki az *e* betűre!

E feladatok közös jellemzője, hogy az eredmény mindig ugyanannyi elemszámú, mint a bemenet volt, s az *i*. tagját a bemenet *i*. tagjából lehet meghatározni. Tehát a bemenő sorozatot le kell másolni, s közben egy -elemre vonatkozó- átalakítást lehet végezni rajta.¹

Bemenet : $N \in \mathbf{N}_0, X \in H^N, f: H \rightarrow G$
 Kimenet : $X \in G^N$
 ef : -
 uf : $\forall i (1 \leq i \leq N) : Y_i = f(X_i)$

Az algoritmus:

Függvény:

$f: H_Elemtípus \rightarrow G_Elemtípus$

Változók:

N : **Egész** [a feldolgozandó sorozat elemei száma]
 X : **Tömb** (1..N:H_Elemtípus) [a feldolgozandó sorozat]
 Y : **Tömb** (1..N:G_Elemtípus) [a feldolgozott sorozat]

A bemenő sorozatból az eredmény elemenként kiszámítható, így nagyon egyszerű megoldást kapunk.

¹ Gyakran e tulajdonságot hívják *elemenkénti feldolgozhatóságnak*. Pontos definíciójával az Adatfeldolgozásról szóló kötetben foglalkozunk.

Másolás (N,X,Y) :
Ciklus I=1-től N-ig
 $Y(I) := f(X(I))$
Ciklus vége
Eljárás vége.

A fejezet elején szereplő példák közül az egyik megoldása:

F23. Magánhangzóra cserélés.

elemek száma: N
 bemenet : X N db elem
 kimenet : Y N db elem
 $f('b')$: 'e', ha magán(b)
 különben 'b'

Másolás (N,X,Y) :
Ciklus I=1-től N-ig
Ha magán(X(I))
akkor $Y(I) := 'e'$
különben $Y(I) := X(I)$
Ciklus vége
Eljárás vége.

2. Kiválogatás

Nem minden esetben kell lemásolni a teljes bemenő sorozatot, hanem annak csupán egy részére kell korlátozni a vizsgálatot. Legegyszerűbb esetben a "vizsgálat" mindössze elemmásolás. Ilyen feladatok szerepelnek e fejezetben.

F24. Egy személyzeti nyilvántartásban emberek neve és személyi száma szerepel, adjuk meg a 20 évnél fiatalabb lányokat!

F25. Adjuk meg egy természetes szám összes osztóját!

F26. Adjuk meg egy osztály azon tanulóit, akik jeles átlagúak!

A feladat részben a keresésre hasonlít, részben pedig a megszámlálásra. Adott tulajdonságú elemet kell megadni, de nem egyet, hanem az összeset; illetve nem megszámlálni kell az adott tulajdonságú elemeket, hanem megadni.

Az eredmény tárolásához egy új tömböt használunk, amelynek elemszámát előre sajnos nem tudjuk megállapítani, csupán egy felső korlátot ismerünk: a lehetséges elemek számát.

Többféle megoldást készíthetünk a feladatra. Az első változatban a keresett elemek sorszámaikat gyűjtjük ki egy vektorba. A megoldás hasonlít a megszámlálásra, csupán a számolás mellett még az elemek sorszámaikat is feljegyezzük. Éppen ezért e változat neve: *kiválogatás kigyűjtéssel*.

Bemenet : $N \in \mathbf{N}_0, X \in H^N$
 $T: H \rightarrow \mathbf{L}, \chi: H \rightarrow \{0, 1\}, \chi(x) = \begin{cases} 1, & \text{ha } T(x) \\ 0, & \text{egyébként} \end{cases}$

Kimenet : $DB \in \mathbf{N}_0$, $Y \in \{1..N\}^N$
 ef : -
 uf : $DB = \sum_{i=1}^N \chi(X_i)$ és $Y \subseteq \{1..N\}$ és $\forall i (1 \leq i \leq DB) : T(X_{Y_i})$

Az algoritmus:

Függvény:

T: Elemtípus \rightarrow Logikai

Változók:

N: **Egész** [a feldolgozandó sorozat elemei száma]
 X: **Tömb**(1..N:Elemtípus) [a feldolgozandó sorozat elemei]
 DB: **Egész** [a megfelelő elemek száma]
 Y: **Tömb**(1..N:Egész) [a megfelelő elemek sorszámai]

Kiválogatás(N,X,DB,Y):

DB:=0

Ciklus I=1-től N-ig

Ha T(X(I)) **akkor** DB:=DB+1: Y(DB):=I

Ciklus vége

Eljárás vége.

Minimális változtatásra lenne szükség, ha nem a sorszámkokat, hanem magukat az elemeket kellene kigyűjteni. Az Y(DB):=I értékadás helyett az Y(DB):=X(I) kell, érdemes a specifikáció átalakulását is meggondolni:

uf : $DB = \sum_{i=1}^N \chi(X_i)$ és $Y \subseteq X$ és $\forall i (1 \leq i \leq DB) : T(Y_i)$

A második változat, a *kiválogatás kiírással* még ennél is egyszerűbb, ebben nincs szükség számolásra, a megtalált elemet azonnal kiírhatjuk. Ez természetesen csak akkor alkalmazható, ha a kiválogatott elemekre további feldolgozás miatt nincs szükség.

Kiválogatás(N,X,DB,Y):

Ciklus I=1-től N-ig

Ha T(X(I)) **akkor** Ki: X(I)

Ciklus vége

Eljárás vége.

A harmadik változatban az elemeket gyűjtjük ki, de -feltételezve, hogy az eredeti sorozatra már nincs szükség- ezeket az eredeti vektorban hagyjuk - ez lesz a *kiválogatás helyben*. Ehhez módosítjuk az utófeltételt.

uf : $DB = \sum_{i=1}^N \chi(X_{be_i})$ és $X_{ki} \subseteq X_{be}$ és $\forall i (1 \leq i \leq DB) : T(X_{ki_i})$

Kiválogatás(N,X,DB,Y):

DB:=0

Ciklus I=1-től N-ig

Ha T(X(I)) **akkor** DB:=DB+1: X(DB):=X(I)

Ciklus vége

Eljárás vége.

Az utolsó változatban sem lesz szükség a kihagyott elemekre, de a megmaradtakat sem akarjuk mozgatni. Emiatt a felesleges elemek helyére egy speciális értéket teszünk, ezzel jelölve a kihagyásukat. Természetesen e megoldás akkor célszerű, ha a későbbi feldolgozások során e speciális érték vizsgálata jóval egyszerűbb, mint maga a kiválogatás volt. E módszer neve: *kiválogatás kiházással*.

Kiválogatás(N,X,DB,Y):

Ciklus I=1-től N-ig

Ha nem T(X(I)) **akkor** X(I):=spec. érték

Ciklus vége

Eljárás vége.

Nézzük meg egy feladat megoldását!

F26. Jeles átlagú tanulók neve.

elemek száma: N
 tanulók : X N db elem
 Rekord(név,átlag)
 jelesszám : DB
 jelek : NEV N db elem

Kiválogatás(N,X,DB,NEV):
 DB:=0
Ciklus I=1-től N-ig
Ha X(I).átlag ≥ 4
akkor DB:=DB+1
 NEV(DB):=X(I).név
Ciklus vége
Eljárás vége.

3. Szétválogatás

Feltehető az előző fejezetbeli feladattípussal kapcsolatban az a kérdés, hogy mi lesz a ki nem válogatott elemekkel. E fejezetben erre a kérdésre adunk választ.

F27. Adott N db természetes szám, válogassuk szét a párosakat és a páratlanokat!

F28. Az osztály tanulóit névsoruk alapján válogassuk szét lányokra, illetve fiúkra!

F29. Az osztály tanulói félévi átlageredményük alapján válogassuk szét jelesekre, jókra, közepesekre, elégségesekre, valamint elégtelenekre!

Ez egy új feladattípusosztály első tagja: itt egy sorozathoz több sorozatot kell rendelni.

E feladatok mindegyike megfogalmazható úgy, hogy végezzünk el egymás után valahány kiválogatást. Az első két feladatban pontosan kettőről van szó, az utolsóban több-

ról, de ez utóbbi megoldható úgy, hogy először válogassuk szét jelesekre és a többiekre, majd a többieket jókra és ... Ezek alapján megállapítható, hogy a szétválogatást elég megfogalmazni két sorozatra, s ha többről van szó, akkor egyszerűen csak többször kell alkalmazni.

A kétféle szétválogatást azonban felesleges két kiválogatással megoldani, ugyanis egyértelmű, hogy amit az első menetben nem válogattunk ki, pontosan azok fognak szerepelni a másodikban.

Az első változatban a kétféle elemet két vektorba válogatjuk szét, ezek elemszáma sajnos mindkét esetben az eredeti vektor elemszáma kell legyen, ugyanis elképzelhető, hogy az összes elemet az egyik vektorba kell majd elhelyezni.

Bemenet : $N \in \mathbb{N}_0, X \in \mathbb{H}^N$

$$T: H \rightarrow L, \chi: H \rightarrow \{0, 1\}, \chi(x) = \begin{cases} 1, & \text{ha } T(x) \\ 0, & \text{egyébként} \end{cases}$$

Kimenet : $DBY, DBZ \in \mathbb{N}_0, Y, Z \in \mathbb{H}^N$

ef : -

$$\text{uf} : DBY = \sum_{i=1}^N \chi(X_i) \text{ és } Y \subseteq X \text{ és } \forall i (1 \leq i \leq DBY) : T(Y_i) \text{ és} \\ DBZ = N - DBY \text{ és } Z \subseteq X \text{ és } \forall i (1 \leq i \leq DBZ) : \neg T(Z_i)$$

Nézzük tehát a szétválogatást két tömbbe!

Az algoritmus:

Függvény:

T: Elemtípus \rightarrow Logikai

Változók:

N: **Egész** [a feldolgozandó sorozat elemei száma]

X: **Tömb**(1..N:Elemtípus) [a feldolgozandó sorozat elemei]

DBY, DBZ: **Egész** [a megfelelő elemek száma]

Y, Z: **Tömb**(1..N:Egész) [a megfelelő elemek sorszámai]

Szétválogatás(N, X, DBY, Y, Z, DBZ):

DBY:=0: DBZ:=0

Ciklus I=1-től N-ig

Ha T(X(I)) **akkor** DBY:=DBY+1: Y(DBY):=X(I)

különben DBZ:=DBZ+1: Z(DBZ):=X(I)

Ciklus vége

Eljárás vége.

Világos, hogy a két vektor alkalmazásával sok felesleges helyet használunk, hiszen $2 \cdot N$ helyet foglalunk pedig összesen csak N-re van szükségünk.

Ha az elemeket egyetlen tömbbe helyezzük el, mégpedig úgy, hogy a T tulajdonságúakat a tömb elejétől, a nem T tulajdonságúakat pedig a végétől kezdve helyezzük el, akkor N helyet megtakaríthatunk. Ez lesz a *szétválogatás egy tömbbe*.

Szétválogatás(N, X, DBY, Y, Z, DBZ):

DBY:=0: INDZ:=N+1

Ciklus I=1-től N-ig

Ha T(X(I)) **akkor** DBY:=DBY+1: Y(DBY):=X(I)

különben INDZ:=INDZ-1: Y(INDZ):=X(I)

Ciklus vége

Eljárás vége.

Vegyük észre, hogy ebben a megoldásban a nem T tulajdonságú elemek sorrendje az eredetihez képest éppen megfordul.

A kiválogatáshoz hasonlóan itt is megoldhatjuk a *helyben szétválogatást* is, ha nincs szükség az elemek eredeti sorrendjére. Ehhez az utófeltételt módosítani kell.

$$\text{uf} : DB = \sum_{i=1}^N \chi(X_{be_i}) \text{ és } X_{ki} = \text{Permutáció}(X_{be}) \text{ és} \\ \forall i (1 \leq i \leq DB) : T(X_i) \text{ és } \forall i (DB < i \leq N) : \neg T(X_i)$$

Egyszerű lenne a következő algoritmus: Elindulunk a tömbben előlről és hátulról, s keresünk olyan elemeket, amelyeket fel kell cserélni. Ha találunk, akkor cserélünk, majd folytatjuk a keresést. Mi ennél egy kicsit hatékonyabb változatot fogunk vizsgálni.

A megoldást úgy végezzük el, hogy a tömb első elemét kivesszük a helyéről. Az utolsó elemtől visszafelé keresünk egy olyat, amely T tulajdonságú, s ezt előre hozzuk a kivett elem helyére. Ezután a hátul felszabadult helyre előlről keresünk egy nem T tulajdonságú elemet, s ha találunk azt hátr tesszük. Mindezt addig végezzük, amíg a tömbben két irányban haladva össze nem találkozunk.

Szétválogatás(N, X, DB):

E:=1: U:=N: **segéd**:=X(E)

Ciklus amíg E<U

Ciklus amíg E<U **és nem** T(X(U))

U:=U-1 [T tul. elem keresése]

Ciklus vége

Ha E<U **akkor** X(E):=X(U): E:=E+1

Ciklus amíg E<U **és** T(X(E))

E:=E+1 [nem T tul. elem keresése]

Ciklus vége

Ha E<U **akkor** X(U):=X(E): U:=U-1

Elágazás vége

Ciklus vége

X(E):=**segéd**

Ha T(X(E)) **akkor** DB:=E **különben** DB:=E-1

Eljárás vége.

Nézzük meg az egyik feladat megoldását, a szétválogatott elemeket egy tömbben elhelyezve előlről, illetve hátulról!

F27. Párosak-páratlanok szétválogatása

elemek száma: N
számok : X N db elem
páros-szám : DB
eredmény : Y N db elem
T(i) : páros(i)

Szétválogatás (N,X,DB,Y) :
DB:=0: IND:=N+1
Ciklus I=1-től N-ig
 Ha páros(X(I))
 akkor DB:=DB+1
 Y(DB):=X(I)
 különben IND:=IND-1
 Y(IND):=X(I)
Ciklus vége
Eljárás vége.

4. Metszet

A következő három feladattípus újabb feladattípusosztályba tartozik, ezekben több sorozathoz kell egyet rendelni.

F30. Adjuk meg két természetes szám osztói ismeretében az összes közös osztójukat!

F31. Nyáron és télen is végeztünk madármegfigyeléseket a Balatonon. Ismerjük, hogy nyáron, illetve télen mely madárfajok fordultak elő. Állapítsuk meg ezek alapján, hogy melyek a nem költöző madarak!

F32. Négy ember heti szabad estéi ismeretében állapítsuk meg, hogy a héten melyik este mehetnek el együtt moziba!

Közös jellemzőjük e feladatoknak, hogy valahány -alapesetben itt is kettő- halmaz elemei közül azokat kell *kiválogatnunk*, amelyek mindegyikben előfordulnak. E halmazok elemeit -a többi programozási tétel tárgyalásmódjához hasonlóan- egy sorozatban (tömbben) felsoroljuk.

Kérdés: Miért nem a sorozatszámítás tételt alkalmazzuk e feladat megoldására?

Válasz: A sorozatszámítás tételben halmazokkal végezhetünk műveleteket, itt pedig olyan sorozatokkal, amelyek halmazelemeket tartalmaznak felsorolva. Emiatt a sorozatszámításban említett, két halmaz közötti metszetműveletet így lehet végrehajtani, ha a halmazokat sorozatként ábrázoljuk.

Bemenet : $N, M \in \mathbb{N}_0, X \in H^N, Y \in H^M, \chi: H \rightarrow \{0, 1\}, \chi(x) = \begin{cases} 1, & x \in Y \\ 0, & \text{egyébként} \end{cases}$
Kimenet : $DB \in \mathbb{N}_0, Z \in H^{\min(N, M)}$

ef : X halmazfelsorolás és Y halmazfelsorolás
uf : $DB = \sum_{i=1}^N \chi(X_i)$ és Z halmazfelsorolás és
 $\forall i (1 \leq i \leq DB): Z_i \in X \text{ és } Z_i \in Y.$

A megoldásban válogassuk ki X olyan elemeit, amelyek benne vannak Y-ban! Az algoritmus tehát egy kiválogatás, amely a belsejében egy eldöntést tartalmaz.

Az algoritmus:

Változók:

N, M: **Egész** [a feldolgozandó sorozat elemei száma]
X, Y: **Tömb**(1..N:ElemTípus) [feldolgozandó sorozatok elemei]
DB: **Egész** [a közös elemek száma]
Z: **Tömb**(1..min(N, M):Egész) [a közös elemek]

Metszet (N, X, M, Y, DB, Z) :

DB:=0 [kiválogatás]
Ciklus I=1-től N-ig
 J:=1
 Ciklus amíg J≤M és X(I)≠Y(J) [eldöntés]
 J:=J+1
 Ciklus vége
 Ha J≤M **akkor** DB:=DB+1: Z(DB):=X(I)
 Ciklus vége
Eljárás vége.

E feladattípus hasonlítható több, az elemi programozási tételek között szerepelt feladattípushoz, ha a feladatot némileg módosítjuk.

Eldöntés: van-e a két halmaznak közös eleme?

Kiválasztás: adjuk meg a két sorozat egyik közös elemét (ha tudjuk, hogy biztosan van ilyen)!

Keresés: ha van, akkor adjuk meg a két sorozat egyik közös elemét!

Megszámolás: hány közös eleme van a két sorozatnak?

Nézzük meg ezek közül például a *keresést*!

Bemenet : $N, M \in \mathbb{N}_0, X \in H^N, Y \in H^M, \chi: H \rightarrow \{0, 1\}, \chi(x) = \begin{cases} 1, & x \in Y \\ 0, & \text{egyébként} \end{cases}$
Kimenet : $VAN \in \mathbb{L}, E \in H$
ef : X halmazfelsorolás és Y halmazfelsorolás
uf : $VAN = \exists i, j (1 \leq i \leq N \text{ és } 1 \leq j \leq M): X_i = Y_j$ és
 $VAN \Rightarrow \exists i, j (1 \leq i \leq N \text{ és } 1 \leq j \leq M): E = X_i = Y_j$

A megoldásban keressünk X-ben egy olyan elemeit, amely benne van Y-ban! Az algoritmus tehát egy keresés, amely a belsejében egy eldöntést tartalmaz.

Változók:

N,M: **Egész** [a feldolgozandó sorozat elemei száma]
 X,Y: **Tömb**(1..N:Elemtípus) [feldolgozandó sorozatok elemei]
 VAN: **Logikai** [van-e közös elem]
 E: Elemtípus [egy közös elem]

Metszetbeli elem(N,X,M,Y,VAN,E):

I:=1: VAN:=hamis

Ciklus amíg I≤N és nem VAN

J:=1

Ciklus amíg J≤M és X(I)≠Y(J)

J:=J+1

Ciklus vége

Ha J≤M **akkor** VAN:=igaz: E:=X(I) **különben** I:=I+1

Ciklus vége

Eljárás vége.

5. Egyesítés (unió)

Ha halmazokról van szó, akkor a metszet mellett természetesen meg kell jelennie az uniónak is.

F33. Két szám prímosztóinak ismeretében adjuk meg legkisebb közös többszörösük prímosztóit!

F34. Egy iskola két földrajztanára órarendjének ismeretében adjuk meg azokat az órákat, amelyben valamelyikük tud egy órát helyettesíteni!

Ebben a feladattípusban tehát azon elemekre vagyunk kíváncsiak, amelyek két halmaz közül legalább az egyikben előfordulnak.

Bemenet : $N, M \in \mathbf{N}_0$, $X \in H^N$, $Y \in H^M$, $\chi: H \rightarrow \{0, 1\}$, $\chi(y) = \begin{cases} 1, & y \in X \\ 0, & \text{egyébként} \end{cases}$

Kimenet : $DB \in \mathbf{N}_0$, $Z \in H^{N+M}$

ef : X halmazfelsorolás és Y halmazfelsorolás

uf : $DB = \sum_{i=1}^N \chi(X_i)$ és Z halmazfelsorolás és

$\forall i (1 \leq i \leq DB): Z_i \in X \text{ vagy } Z_i \in Y.$

A megoldásban másoljuk le X elemeit Z-be, majd válogassuk ki Y olyan elemeit, amelyek nincsenek benne X-ben! Az algoritmus tehát egy másolás, majd egy kiválogatás, amely a belsejében egy eldöntést tartalmaz.

Változók:

N,M: **Egész** [a feldolgozandó sorozat elemei száma]
 X,Y: **Tömb**(1..N:Elemtípus) [feldolgozandó sorozatok elemei]
 DB: **Egész** [a közös elemek száma]
 Z: **Tömb**(1..N+M:Egész) [a közös elemek]

Egyesítés(N,X,M,Y,DB,Z):

Z:=X: DB:=N [másolás]

Ciklus J=1-től M-ig [kiválogatás]

I:=1

Ciklus amíg I≤N és X(I)≠Y(J) [eldöntés]

I:=I+1

Ciklus vége

Ha I>N **akkor** DB:=DB+1: Z(DB):=X(I)

Ciklus vége

Eljárás vége.

A példák megoldása helyett itt is egy alkalmazást nézzünk: egy sorozatból készítsünk halmazfelsorolást! A feladat tehát az, hogy másoljuk le egy sorozat elemeit, de az azonos értékű elemek az eredményben csak egyszer szerepeljenek.

Ez egy furcsa egyesítés, $Z = Z \cup X$ képlettel írhatnánk le, azaz azokat az elemeket vegyük bele X-ből az eredménybe, amelyek még nem szerepelnek benne.

Halmazfelsorolás_készítés(N,X,DB,Z):

DB:=0

Ciklus I=1-től N-ig

J:=1

Ciklus amíg J≤DB és X(I)≠Z(J)

J:=J+1

Ciklus vége

Ha J>DB **akkor** DB:=DB+1: Z(DB):=X(I)

Ciklus vége

Eljárás vége.

6. Összefuttatás (rendezett uniója)

Bizonyos esetekben az unió feladattípus megoldása az előző fejezetbelinél hatékonyabb lehet. Ezt is konkrét feladatokon keresztül vizsgáljuk.

F35. Egy osztály lány-, illetve fiútanulóinak névsora alapján állítsuk elő az osztálynévsort!

F36. Egy iskolában négy szakkörre járnak tanulók (van aki többre is). A szakkörnévsorok alapján állítsuk elő a szakkörre járó tanulók névsorát!

Megállapíthatjuk, hogy az általános egyesítéshez képest itt az a specialitás, hogy mindegyik sorozatként ábrázolt halmaz rendezett, s az eredménynek is rendezettnek kell lenni.

Ha nem két sorozatról van szó, akkor az a korábbiaknak megfelelően visszavezethető két sorozat feldolgozására.

Bemenet : $N, M \in \mathbf{N}_0$, $X \in H^N$, $Y \in H^M$, $\chi: H \rightarrow \{0, 1\}$, $\chi(y) = \begin{cases} 1, & y \notin X \\ 0, & \text{egyébként} \end{cases}$

Kimenet : $DB \in \mathbf{N}_0$, $Z \in H^{N+M}$

ef : X halmazfelsorolás és Y halmazfelsorolás és X rendezett és Y rendezett

uf : $DB = \sum_{i=1}^N \chi(X_i)$ és Z halmazfelsorolás és Z rendezett és $\forall i (1 \leq i \leq DB): Z_i \in X \text{ vagy } Z_i \in Y$.

A megoldásban haladjunk párhuzamosan a két sorozatban! Az eredmény első eleme vagy X_I , vagy Y_J lesz. Amelyik kisebb, azt az eredményssorozatba tesszük, abban a sorozatban kell továbblépni egy elemmel, s újra egy-egy elemet hasonlítani. Ha egyenlők voltak, akkor az egyiket másoljuk az eredménybe, majd mindkét sorozatban továbblépünk. Ha az egyiknek a végére értünk, akkor a másikat minden változtatás nélkül az eredménybe másoljuk.

Változók:

N, M : **Egész** [a feldolgozandó sorozat elemei száma]
 X, Y : **Tömb**(1..N:Elemtípus) [feldolgozandó sorozatok elemei]
 DB : **Egész** [a közös elemek száma]
 Z : **Tömb**(1..N+M:Egész) [a közös elemek]

Összefuttatás (N, X, M, Y, DB, Z):

I:=1: J:=1: DB:=0

Ciklus amíg I≤N és J≤M

DB:=DB+1

Elágazás

X(I)<Y(J) **esetén** Z(DB):=X(I): I:=I+1

X(I)=Y(J) **esetén** Z(DB):=X(I): I:=I+1: J:=J+1

X(I)>Y(J) **esetén** Z(DB):=Y(J): J:=J+1

Elágazás vége

Ciklus vége

Ciklus amíg I≤N

DB:=DB+1: Z(DB):=X(I): I:=I+1

Ciklus vége

Ciklus amíg J≤M

DB:=DB+1: Z(DB):=Y(J): J:=J+1

Ciklus vége

Eljárás vége.

Észrevehetjük a megoldást elemezve, hogy ha olyan szerencsénk volt, hogy $X(N)=Y(M)$, akkor az utolsó két ciklusra tulajdonképpen nincs is szükség, hiszen nem maradt másolnivaló. Az a baj, hogy ez a szerencsés eset viszonylag ritkán fordul elő.

A második megoldásváltozatban mi magunk idézzük elő e szerencsét: mindkét sorozat végére egy nagyon nagy (az adott típus legnagyobb értéke), de egyező elemet teszünk.

Összefuttatás (N, X, M, Y, DB, Z):

I:=1: J:=1: DB:=0

X(N+1):=+∞: Y(M+1):=+∞ [az elemtípus maximális értéke]

Ciklus amíg I<N+1 vagy J<M+1

DB:=DB+1

Elágazás

X(I)<Y(J) **esetén** Z(DB):=X(I): I:=I+1

X(I)=Y(J) **esetén** Z(DB):=X(I): I:=I+1: J:=J+1

X(I)>Y(J) **esetén** Z(DB):=Y(J): J:=J+1

Elágazás vége

Ciklus vége

Eljárás vége.

Ebben a megoldásban a hozzávett fiktív elem nem kerül be az eredménybe. Ha szükségünk lenne rá, a ciklus vége után még elhelyezhetnénk az eredmény végére.

A harmadik változatban kihasználjuk azt a -nem minden feladatban meglevő- specialitást, hogy a két sorozatban biztosan nincs közös elem. Ilyen például a fejezet elején szereplő F35 feladat. Ezt az altípust a megkülönböztetés érdekében **összefésülésnek** nevezzük.

Összefésülés (N, X, M, Y, DB, Z):

I:=1: J:=1: DB:=0

X(N+1):=+∞: Y(M+1):=+∞ [az elemtípus maximális értéke]

Ciklus amíg I<N+1 vagy J<M+1

DB:=DB+1

Ha X(I)<Y(J) **akkor** Z(DB):=X(I): I:=I+1

különben Z(DB):=Y(J): J:=J+1

Ciklus vége

Eljárás vége.

III. Rendezések

A sorozathoz sorozatot rendelő feladattípus osztály egyik tagját külön vizsgáljuk a fejezetben.

Az alapfeladat egy N elemű sorozat nagyság szerinti sorba rendezése. A sorozat elemei olyanok, amelyekre a \leq relációk léteznek. Feltesszük a feladatok mindegyikében, hogy a sorozathoz létezik indexelés művelet, s ezt a megoldásban ki is használjuk. Az eredmény a módszerek jelentős részében helyben keletkezik, az eredeti sorrend elvész.

Konkrét feladatok ismertetése és megoldása helyett ebben a fejezetben egy számpéldát fogunk végignézni az egyes módszereknél. E számsorozat -egy kivétellel- a következő lesz: 5, 3, 9, 1, 7.

Bemenet : $N \in \mathbb{N}_0, X \in H^N$

Kimenet : $X \in H^N$

ef : -

uf : X_{ki} rendezett és $X_{ki} = \text{Permutáció}(X_{be})$.

Az algoritmus:

Változók:

N : **Egész** [a feldolgozandó sorozat elemei száma]

X : **Tömb** (1..N:Elem típus) [a feldolgozandó sorozat]

Az egyes módszereket összehasonlítjuk tárigény (a rendezésben résztvevő tároló helyek száma), valamint végrehajtási idő (hasonlítások száma, mozgatók száma) szerint. A helyfoglalásba nem számítjuk bele a ciklusváltozókat, a végrehajtási időbe pedig ezek növelését, vizsgálatát, hiszen ezek mérete és ideje nem függ a rendezendő elemek típusától.

A hasonlítások és a mozgatók száma néha nem függ a rendezendő elemek értékétől, a legtöbb esetben azonban igen, így csak minimális és maximális számokról beszélhetünk.

1. Egyszerű cserés rendezés

Az első módszer a következő ötletre épül. Hasonlítsuk össze az első elemet a sorozat összes többi mögötte levő elemével, s ha valamelyik kisebb nála, akkor cseréljük meg

azzal! Ezzel elérhetjük, hogy a sorozat első helyére a legkisebb elem kerül. Folytassuk ugyanezen elven a sorozat második elemével, utoljára pedig az utolsóelőttivel.

Rendezés (N, X) :

Ciklus $I=1$ -től $N-1$ -ig

Ciklus $J=I+1$ -től N -ig

Ha $X(I) > X(J)$ **akkor** $\text{Csere}(X(I), X(J))$

Ciklus vége

Ciklus vége

Eljárás vége.

A belső ciklus ciklusfeltétele kiértékelése előtt a sorozat:

$I=1 \Rightarrow 5, 3, 9, 1, 7$

3, 5, 9, 1, 7

3, 5, 9, 1, 7

1, 5, 9, 3, 7

1, 5, 9, 3, 7

$I=2 \Rightarrow 1, 5, 9, 3, 7$

1, 5, 9, 3, 7

1, 3, 9, 5, 7

1, 3, 9, 5, 7

$I=3 \Rightarrow 1, 3, 9, 5, 7$

1, 3, 5, 9, 7

1, 3, 5, 9, 7

$I=4 \Rightarrow 1, 3, 5, 9, 7$

1, 3, 5, 7, 9

Helyfoglalás: $N+1$

Hasonlítások száma: $N*(N-1)/2$ $O(n^2)^1$

Mozgatók száma: $0 - 3*N*(N-1)/2$ $O(n^2)$
[a csere mindig 3 mozgató]

2. Minimumkiválasztásos rendezés

Az előző -egyszerű- módszer hibája a sok felesleges csere. Célszerűbb lenne az aktuális első elemet a mögötte levők közül egyedül a legkisebbel cserélni. Ehhez a rendező ciklus belsejében cserék helyett egy minimumkiválasztást kell csinálni.

Rendezés (N, X) :

Ciklus $I=1$ -től $N-1$ -ig

$\text{MIN} := I$

Ciklus $J=I+1$ -től N -ig

Ha $X(\text{MIN}) > X(J)$ **akkor** $\text{MIN} := J$

Ciklus vége

$\text{Csere}(X(I), X(\text{MIN}))$

Ciklus vége

Eljárás vége.

Itt a számpéldát a külső ciklus ciklusfeltételének kiértékelése előtt vizsgáljuk:

¹ A nagy ordó(x) a matematikában szokásos jelölés, jelentése: valamilyen jellemző értékét kifejező polinom legmagasabb hatványkitevőjű tagja x-szel arányos, pl. $O(x^2)ax^2+bx+c$ típusú mennyiséget jelöl.

$I=1 \Rightarrow 5, 3, 9, 1, 7$
 $I=2 \Rightarrow 1, 3, 9, 5, 7$
 $I=3 \Rightarrow 1, 3, 9, 5, 7$
 $I=4 \Rightarrow 1, 3, 5, 9, 7$

Helyfoglalás: $N+1$

Hasonlítások száma: $N*(N-1)/2$ $O(n^2)$

Mozgatások száma: $3*(N-1)$ $O(n)$

Érdekessége e módszernek az, hogy a mozgatók száma szerencsétlen esetben rosszabb lehet, mint az előzőé, amelynek javításaként keletkezett. Ez amiatt van, hogy itt a külső ciklusban mindenképpen cserélünk. Meg lehetne ugyan vizsgálni, hogy érdemes-e cserélni, de ez a vizsgálat a legtöbb esetben növelné a futási időt.

3. Buborékos rendezés

Most egy másik rendezési alapelvet vizsgálunk meg: hasonlítsuk egymással a szomszédos elemeket, s ha a sorrendjük nem jó, akkor cseréljük meg őket!

Ezzel a módszerrel egy cikluslépés lefutása alatt a legnagyobb elem biztosan a sorozat végére kerül, s ezen kívül a nagyobb értékű elemek hátrafelé, a kisebbek pedig előre felé mozdulnak el (innen van a buborékmódszer elnevezés).

Rendezés (N, X):

Ciklus $I=N-tól$ $2-ig$ $-1-esével$

Ciklus $J=1-tól$ $I-1-ig$

Ha $X(J) > X(J+1)$ **akkor** $Csere(X(J), X(J+1))$

Ciklus vége

Ciklus vége

Eljárás vége.

Újra a belső ciklusnál nézzük a konkrét rendezési példát:

$I=5 \Rightarrow 5, 3, 9, 1, 7$
 $3, 5, 9, 1, 7$
 $3, 5, 9, 1, 7$
 $3, 5, 1, 9, 7$
 $3, 5, 1, 7, 9$
 $I=4 \Rightarrow 3, 5, 1, 7, 9$
 $3, 5, 1, 7, 9$
 $3, 1, 5, 7, 9$
 $3, 1, 5, 7, 9$
 $I=3 \Rightarrow 3, 1, 5, 7, 9$
 $1, 3, 5, 7, 9$
 $1, 3, 5, 7, 9$
 $I=2 \Rightarrow 1, 3, 5, 7, 9$
 $1, 3, 5, 7, 9$

Helyfoglalás: $N+1$

Hasonlítások száma: $N*(N-1)/2$

Mozgatások száma: $0 - 3*N*(N-1)/2$

$O(n^2)$

$O(n^2)$

4. Javított buborékos rendezés

Az előző módszer azért nem váltotta be reményeinket, mert nem használtuk ki benne, hogy az elemek a helyük felé elmozdultak. A legegyszerűbb esetben például, ha a belső ciklus egy lefutása alatt egyáltalán nem volt csere, akkor felesleges minden további hasonlítás, a rendezéssel készen vagyunk. Ha volt csere, de például a legutolsó a sorozat közepénél volt, akkor ebből tudhatjuk, hogy a sorozat a közepe után biztosan kész, rendezett, s csak az előtte levő résszel kell foglalkozni.

Figyeljük tehát minden menetben a legutolsó csere helyét, s a következő menetben csak addig rendezzünk! (Át kell térni 'amíg'-os ciklusra.)

Rendezés (N, X):

$I:=N$

Ciklus **amíg** $I \geq 2$

$CS:=0$

Ciklus $J=1-tól$ $I-1-ig$

Ha $X(J) > X(J+1)$ **akkor** $Csere(X(J), X(J+1))$: $CS:=J$

Ciklus vége

$I:=CS$

Ciklus vége

Eljárás vége.

Az előző módszer számpéldájából sorok maradnak ki:

$I=5 \Rightarrow 5, 3, 9, 1, 7$
 $3, 5, 9, 1, 7$
 $3, 5, 9, 1, 7$
 $3, 5, 1, 9, 7$
 $3, 5, 1, 7, 9$
 $I=4 \Rightarrow 3, 5, 1, 7, 9$
 $3, 5, 1, 7, 9$
 $3, 1, 5, 7, 9$
 $3, 1, 5, 7, 9$
 $I=2 \Rightarrow 1, 3, 5, 7, 9$
 $1, 3, 5, 7, 9$

Helyfoglalás: $N+1$

Hasonlítások száma: $N-1 - N*(N-1)/2$

Mozgatások száma: $0 - 3*N*(N-1)/2$

$O(n^2)$

$O(n^2)$

A hasonlítások számában javulást látunk, az igazi javulás azonban nem a minimális és a maximális, hanem az átlagos végrehajtási időben van.

5. Beillesztéses rendezés

Újabb rendezési elvvel ismerkedünk meg. Az eddigi módszerek mindegyike olyan volt, hogy a sorozatot felosztotta egy már kész, rendezett szakaszra, s a rendezést a másik szakasz elemei között folytatta. Másik jellemzőjük volt, hogy a rendezés elkezdéséhez már az összes elemnek rendelkezésre kellett állnia.

Most a kártyakeveréshez hasonló elvből indulunk ki: egyetlen elem mindig rendezett; ha van egy rendezett részsorozatunk, akkor abba a nagyság szerinti helyére illesszük be a soron következő elemet!

Rendezés (N,X) :

Ciklus I=2-től N-ig

J:=I-1

Ciklus amíg J>0 és X(J)>X(J+1)

Csere(X(J),X(J+1)) : J:=J-1

Ciklus vége

Ciklus vége

Eljárás vége.

A számpélda a belső ciklus feltételénél itt a következőképpen alakul:

I=2 ⇒ 5, 3, 9, 1, 7
 3, 5, 9, 1, 7
 I=3 ⇒ 3, 5, 9, 1, 7
 I=4 ⇒ 3, 5, 9, 1, 7
 3, 5, 1, 9, 7
 3, 1, 5, 9, 7
 1, 3, 5, 9, 7
 I=5 ⇒ 1, 3, 5, 9, 7
 1, 3, 5, 7, 9

Helyfoglalás: N+1

Hasonlítások száma: N-1 - N*(N-1)/2

$O(n^2)$

Mozgatások száma: 0 - 3*N*(N-1)/2

$O(n^2)$

6. Javított beillesztéses rendezés

Észrevehetjük az előző módszernél, hogy a beillesztő ciklusban a legtöbb elem egyszer mozdul el, a beillesztendő viszont sokszor. Ezt a sok mozgást is csökkenthetjük egyetlenegyre úgy, hogy a beillesztendőt nem tesszük azonnal be a sorozatba, hanem csak a többieket tologatjuk hátra, s a beillesztendőt csak a végén tesszük a helyére.

Rendezés (N,X) :

Ciklus I=2-től N-ig

J:=I-1 : Y:=X(I)

Ciklus amíg J>0 és X(J)>Y

X(J+1):=X(J) : J:=J-1

Ciklus vége

X(J+1):=Y

Ciklus vége

Eljárás vége.

Az elemek mozgatása miatt e példában a kivett elem visszahelyezéskéig egyes elemek kétszer szerepelnek.

I=2 ⇒ 5, 3, 9, 1, 7
 5, 5, 9, 1, 7
 I=3 ⇒ 3, 5, 9, 1, 7
 I=4 ⇒ 3, 5, 9, 1, 7
 3, 5, 1, 9, 7

Helyfoglalás: N+1

Hasonlítások száma: N-1 - N*(N-1)/2

$O(n^2)$

Mozgatások száma: 2*(N-1) - 2*(N-1)+N*(N-1)/2

$O(n^2)$

3, 1, 5, 9, 7
 3, 3, 5, 9, 7
 I=5 ⇒ 1, 3, 5, 9, 7
 1, 3, 5, 9, 9

7. Szétesztő rendezés

A három alaprendezés, s azok javításai után speciális rendezésekkel foglalkozunk, amelyekben előfeltételként speciális megszorításaink lesznek.

A legelsőben feltesszük, hogy a rendezendő elemek olyan rekordok, amelyek kulcsmezője (vagy egy abból kiszámolt számérték) 1 és N közötti természetes szám lehet, s nincs két azonos kulcsú rekord.

A kulcsmező itt egyértelműen megadja azt a helyet, ahova az elemet tenni kell, így semmi hasonlításra nincs szükség. A módszerhez azonban egy újabb tömbre van szükség, ahova az eredményt elhelyezzük.

Bemenet : $N \in \mathbb{N}_0$, $X \in H^N$, $H = (\text{kulcs}, \text{egyéb})$

Kimenet : $Y \in H^N$

ef : $X.\text{kulcs} = \text{Permutáció}([1, \dots, N])$

uf : Y rendezett és $Y = \text{Permutáció}(X)$.

Az algoritmus:

Változók:

N : **Egész** [a feldolgozandó sorozat elemei száma]

X, Y : **Tömb**(1..N:H_Elemtípus) [a két sorozat]

Rendezés (N,X,Y) :

Ciklus I=1-től N-ig

Y(X(I).kulcs) := X(I)

Ciklus vége

Eljárás vége.

Itt a speciális feltétel miatt meg kell változtatnunk a példaszorozatot: 3, 2, 5, 1, 4 (most csak a kulcsmező értékét tároljuk). A feladat másik specialitása miatt pedig két sorozatot kell adnunk, a bemenetet és az eredményt. Az eredmény még nem kitöltött tagjait jelöli.

Bemenet: 3, 2, 5, 1, 4

Kimenet: I=1 ⇒ ⊗, ⊗, ⊗, ⊗, ⊗

I=2 ⇒ ⊗, ⊗, 3, ⊗, ⊗

I=3 ⇒ ⊗, 2, 3, ⊗, ⊗

I=4 ⇒ ⊗, 2, 3, ⊗, 5

I=5 ⇒ 1, 2, 3, ⊗, 5

I=6 ⇒ 1, 2, 3, 4, 5

Helyfoglalás: 2*N

Hasonlítások száma: 0

Mozgatások száma: N

Kulcsmezőindexelés: N

8. Számlálva szétosztó rendezés

Egy kicsit kevesebbet követel előfeltételként a következő rendezés: itt az elemek kulcsmezője lehet az $[1, N]$ -nél szélesebb intervallumban is és nem szükséges feltétel a kulcsok különbözősége.

Bemenet : $N, M \in \mathbb{N}_0, X \in H^N, H = (\text{kulcs}, \text{egyéb})$
 Kimenet : $Y \in H^N$
 ef : $\forall i \ (1 \leq i \leq N): X_i.\text{kulcs} \in \{1..M\}$
 uf : Y rendezett és $Y = \text{Permutáció}(X)$.

Itt első lépésként számoljuk le minden lehetséges kulcsértékre, hogy hány ilyen értékű elem van! Mivel a kulcsértékekkel indexelhetünk, ezért a számlálás egyetlen indexeléssel elvégezhető minden elemre.

Másodjára minden lehetséges kulcsértékhez határozzuk meg a nála kisebb vagy egyenlő kulcsú rekordok számát!

Harmadik lépésként minden elemet közvetlenül a helyére helyezhetünk a fenti információ alapján.

Rendezés (N, X) :
 DB ($1..M$) := (0, ..., 0)
Ciklus $I = 1$ -től N -ig
 DB ($X(I).\text{kulcs}$) := DB ($X(I).\text{kulcs}$) + 1
Ciklus vége
Ciklus $J = 2$ -től M -ig
 DB (J) := DB (J) + DB ($J-1$)
Ciklus vége
Ciklus $I = 1$ -től N -ig
 $Y(\text{DB}(X(I).\text{kulcs})) := X(I)$
 DB ($X(I).\text{kulcs}$) := DB ($X(I).\text{kulcs}$) - 1
Ciklus vége
Eljárás vége.

Itt a speciális rendezés miatt újabb példaszorozatot vizsgálunk: 5, 3, 5, 1, 7. A bemenet mellett először a DB vektor értékeit közöljük, majd az eredményt.

Bemenet:
 5, 3, 5, 1, 7

DB:
 $I=1 \Rightarrow 0, 0, 0, 0, 0, 0$
 $I=2 \Rightarrow 0, 0, 0, 0, 1, 0, 0$
 $I=3 \Rightarrow 0, 0, 1, 0, 1, 0, 0$
 $I=4 \Rightarrow 0, 0, 1, 0, 2, 0, 0$
 $I=5 \Rightarrow 1, 0, 1, 0, 2, 0, 0$
 $I=6 \Rightarrow 1, 0, 1, 0, 2, 0, 1$

A második ciklusban:

DB:	Kimenet:
$J=2 \Rightarrow 1, 0, 1, 0, 2, 0, 1$	$I=1 \Rightarrow \otimes, \otimes, \otimes, \otimes, \otimes$
$J=3 \Rightarrow 1, 1, 1, 0, 2, 0, 1$	$I=2 \Rightarrow \otimes, \otimes, \otimes, 5, \otimes$
$J=4 \Rightarrow 1, 1, 2, 0, 2, 0, 1$	$I=3 \Rightarrow \otimes, 3, \otimes, 5, \otimes$
$J=5 \Rightarrow 1, 1, 2, 2, 2, 0, 1$	$I=4 \Rightarrow \otimes, 3, 5, 5, \otimes$
$J=6 \Rightarrow 1, 1, 2, 2, 4, 0, 1$	$I=5 \Rightarrow 1, 3, 5, 5, \otimes$
$J=7 \Rightarrow 1, 1, 2, 2, 4, 4, 1$	$I=6 \Rightarrow 1, 3, 5, 5, 7$
1, 1, 2, 2, 4, 4, 5	

Helyfoglalás: $2*N+M*\varepsilon$

Hasonlítások száma: 0

Mozgatások száma: N

Kulcsmezőindexelés: $5*N$

9. Számláló rendezés

Az egyszerű cserés módszer mozgatasainak számát is javíthatjuk az egyik újabb módszer, a szétosztó rendezés segítségével. A cserés rendezésnél ne cserélgessük az elemeket, hanem csak számoljuk meg mindegyikre, hogy hány nála kisebb, illetve az öt megelőzők között hány vele egyenlő van (magát is beleértve). Ezzel a rendezendő adatsorhoz a $[0, N-1]$ intervallum egész számainak egy permutációját rendeltük. Ez a permutáció lehet a bemenete a szétosztó rendezésnek.

Rendezés (N, X) :
 DB ($1..N$) := 1 [a tömb 1-es értékekkel feltöltése]
Ciklus $I = 1$ -től N -ig
 Ciklus $J = I+1$ -től N -ig
 Ha $X(I) > X(J)$ akkor DB (I) := DB (I) + 1
 különben DB (J) := DB (J) + 1
 Ciklus vége
 Ciklus vége
 Ciklus $I = 1$ -től N -ig
 $Y(\text{DB}(I)) := X(I)$ [szétosztó rendezés]
 Ciklus vége
Eljárás vége.

Újra az eredeti számsorozatot vizsgáljuk (5, 3, 9, 1, 7):

A DB vektor értékei:

$I=1 \Rightarrow 1, 1, 1, 1, 1$
 $I=2 \Rightarrow 3, 1, 2, 1, 2$
 $I=3 \Rightarrow 3, 2, 3, 1, 2$
 $I=4 \Rightarrow 3, 2, 3, 1, 3$
 $I=5 \Rightarrow 3, 2, 3, 1, 3$

Helyfoglalás: $2*N+N*\varepsilon$

Hasonlítások száma: $N*(N-1)/2$

Mozgatások száma: N

2-szeres indexelés: N

Az eredmény:

$I=1 \Rightarrow \otimes, \otimes, \otimes, \otimes, \otimes$
 $I=2 \Rightarrow \otimes, \otimes, 5, \otimes, \otimes$
 $I=3 \Rightarrow \otimes, 3, 5, \otimes, \otimes$
 $I=4 \Rightarrow \otimes, 3, 5, \otimes, 9$
 $I=5 \Rightarrow 1, 3, 5, \otimes, 9$
 $I=6 \Rightarrow 1, 3, 5, 7, 9$

```

3, 2, 4, 1, 3
3, 2, 5, 1, 3
I=4 ⇒ 3, 2, 5, 1, 3
3, 2, 5, 1, 4

```

10. Rendezés Shell módszerrel

Sokat javíthat a rendező módszeren, ha először a nagy távolságú elemeket hasonlítjuk és cseréljük, mert így az egyes elemek nagyon gyorsan közel kerülhetnek a végleges helyükhöz. Ha egy rendezés ezt az információt kihasználja, akkor az eredeténél lényegesen jobbat kaphatunk. Ilyen módszer például a beillesztéses rendezés.

A használt lépésköz többféle lehet, közös jellemzőjük, hogy szigorúan monoton csökkenők, s utoljára éppen 1 az értékük.

Rendezés (N,X) :

L:=L0

Ciklus amíg L≥1 [lépésköz-ciklus]

Ciklus K=L+1-től 2*L-ig [K: az L. részsorozat 2. eleme]

Ciklus I=K-től N-ig L-esével

J:=I-L: Y:=X(I) [javított beillesztéses rendezés]

Ciklus amíg J>0 és X(J)>Y

X(J+L):=X(J): J:=J-L

Ciklus vége

X(J+L):=Y

Ciklus vége

Ciklus vége

L:=Következő(L)

Ciklus vége

Eljárás vége.

Az L értékének meghatározására többféle módszer van. Lehetséges, hogy 2^{k-1} alakú, lehetséges, hogy Fibonacci szám alakú. Kezdőértékként mindenképpen olyan értéket kell meghatározni, amely nem lépi túl az N értékét, majd a **Következő** függvény így alakul:

Következő (L) : [2^{k-1} alakú esetben]

Következő:=(L+1)/2-1

Eljárás vége.

Következő (L) : [Fibonacci szám alakú esetben²]

L2:=L-L1: L:=L1: L1:=L2: Következő:=L1

Eljárás vége.

² Ha L1 és L egymás utáni Fibonacci számok, akkor L2 az előző Fibonacci szám lesz. A módszernél nyilván előre adott az L0 (kezdő L) és az L1 értéke.

IV. Keresések

Ebben a fejezetben kétféle speciális keresési feladattal foglalkozunk. Az elsőben egy adott értékű elem megkeresése a feladat egy rendezett sorozatban, a másodikban pedig N db sorozatban kell keresnünk egy-egy értéket úgy, hogy azok egymásnak valamilyen szempont szerint megfeleljenek.

1. Keresés rendezett sorozatban

Ebben az esetben tehát adott egy rendezett sorozat, amelyben egy adott értékű elem sorszámát kell meghatározni, ha az benne van a sorozatban.

Bemenet : $N \in \mathbf{N}_0$, $X \in \mathbf{H}^N$, $Y \in \mathbf{H}$, $T: \mathbf{H} \rightarrow \mathbf{L}$

Kimenet : $\text{VAN} \in \mathbf{L}$, $\text{SORSZ} \in \mathbf{N}$

ef : Rendezett(X)

uf : $\text{VAN} \equiv (\exists i (1 \leq i \leq N): X_i = Y)$ és
 $\text{VAN} \Rightarrow 1 \leq \text{SORSZ} \leq N$ és $X_{\text{SORSZ}} = Y$

Az algoritmus:

Változók:

N: **Egész** [a feldolgozandó sorozat elemei száma]

X: **Tömb**(1..N:Elemtípus) [a feldolgozandó sorozat elemei]

Y: Elemtípus [a keresett elem]

VAN: **Logikai** [az eredmény - van-e megfelelő elem]

SORSZ: **Egész** [az eredmény - a megfelelő elem sorszáma]

Próbáljuk először a *lineáris keresést* alkalmazni e feladat megoldására! A feladat specialitásából következik, hogy ha egy, a keresett elemnél nagyobb értékű elemet találunk, akkor a keresett elem már biztosan nem fordulhat elő, a keresést be lehet fejezni. Emiatt a program utófeltétele a következő lesz:

uf : $\text{VAN} \equiv (\exists i (1 \leq i \leq N): X_i = Y)$ és $\text{VAN} \Rightarrow 1 \leq \text{SORSZ} \leq N$ és
 $X_{\text{SORSZ}} = Y$ és $\forall i (1 \leq i < \text{SORSZ}): X_i < Y$

Keresés (N,X,Y,VAN,SORSZ) :

I:=1

Ciklus amíg I≤N és X(I)<Y

I:=I+1

Ciklus vége

VAN:=(I≤N) és X(I)=Y

Ha VAN **akkor** SORSZ:=I

Eljárás vége.

Nézzük meg, hogy e keresést milyen hasonlítás-szám jellemez!

Minimum: 1
Átlag: $(N+1)/2$
Maximum: N

Az átlagos hasonlítás-számban szereplő lineáris összefüggés miatt hívják e keresést *lineáris keresésnek*.

Logaritmikus keresés

A rendezettséget (és az indexelhetőséget) másképpen is kihasználhatjuk. Vizsgáljuk meg első lépésben a sorozat középső elemét! Ha ez a keresett elem, akkor készen vagyunk. Ha a keresett elem ennél kisebb, akkor csak az ezt megelőzők között lehet, tehát a keresést a továbbiakban a sorozatnak arra a részére kell alkalmazni. Ha a keresett ennél nagyobb, akkor pedig ugyanezen elv alapján a sorozat ezt követő részére.

Ez a megoldás szintén az utófeltételt módosítja, szigorítja. Itt az előzővel ellentétben nem lehet tudni, hogy a megtalált érték az azonos értékek közül az első, az utolsó, vagy pedig valamelyik középső lesz.

uf : $VAN \equiv (\exists i (1 \leq i \leq N) : X_i = Y)$ és $VAN \Rightarrow 1 \leq SORSZ \leq N$ és $X_{SORSZ} = Y$ és $\forall i (1 \leq i < SORSZ) : X_i \leq Y$ és $\forall i (SORSZ < i \leq N) : X_i \geq Y$

Keresés (N, X, Y, VAN, SORSZ) :

E:=1: U:=N

Ciklus

K:=[(E+U)/2] [E+U felének egész értéke]

Elágazás

Y<X(K) esetén U:=K-1

Y>X(K) esetén E:=K+1

Elágazás vége

amíg E≤U és X(K)≠Y

Ciklus vége

VAN:=(E≤U)

Ha VAN **akkor** SORSZ:=K

Eljárás vége.

Minimum: 1

Átlag: $\log_2(N)$

Maximum: $\log_2(N)+1$

Az átlagos hasonlítás-számban szereplő logaritmikus összefüggés miatt hívják e keresést *logaritmikus keresésnek*, illetve a sorozat felezésére utalva *felezéses* vagy *bináris keresésnek*.

Ennek a keresésnek is meg lehet alkotni a rokonait, mint azt tettük a lineáris kereséssel, azaz készíthetünk belőle **eldöntést, kiválasztást, megszámlást, kiválogatást**. Ezek közül nézzük meg a kiválogatást!

A módosított specifikáció:

Bemenet : $N \in \mathbb{N}_0, X \in H^N, Y \in H, T: H \rightarrow L$

Kimenet : $VAN \in L, SORSZ \in \mathbb{N}, E, U \in \mathbb{N}$

ef : Rendezett (X)

uf : $VAN \equiv (\exists i (1 \leq i \leq N) : X_i = Y)$ és

$VAN \Rightarrow 1 \leq E \leq U \leq N$ és $\forall i (E \leq i \leq U) : X_i = Y$

Nyilvánvaló, hogy az azonos értékű elemek itt mindenképpen egymás mellett fognak szerepelni, azaz a kiválogatáshoz elég megadni az első, illetve az utolsó ilyen elem sorszámát. Ezt úgy tesszük meg, hogy a logaritmikus keresés után előre, illetve hátrafelé megkeressük az utolsó megfelelő elemet.

Kiválogatás (N, X, Y, VAN, E, U) :

E:=1: U:=N

Ciklus

K:=[(E+U)/2]

Elágazás

Y<X(K) esetén U:=K-1

Y>X(K) esetén E:=K+1

Elágazás vége

amíg E≤U és X(K)≠Y

Ciklus vége

VAN:=(E≤U)

Ha VAN **akkor** E:=K

Ciklus amíg E>1 és X(E-1)=Y

E:=E-1

Ciklus vége

U:=K

Ciklus amíg U<N és X(U+1)=Y

U:=U+1

Ciklus vége

Elágazás vége

Eljárás vége.

2. Visszalépéses keresés (backtrack)

Újra konkrét feladatokkal kezdjük e feladattípus ismertetését.

F37. Helyezzünk el egy sakktáblán 8 vezért úgy, hogy egyik se üsse a másikat!

F38. Egy vállalat N db munkára szeretne munkásokat felvenni. Jelentkezik N db munkás, mindegyik megmondja, hogy milyen munkát tudna elvégezni. Osszuk el közöttük a munkákat úgy, hogy mindenkinek jusson munka, s minden munkát elvégezzenek!

F39. N db közért M db pékségtől rendelhet kenyeret. Ismerjük a közértek kenyérigényét, a pékségek sütési kapacitását, valamint azt, hogy melyik közért mely pékségekkel áll kapcsolatban. Adjuk meg, hogy melyik közért melyik pékségtől rendelje a kenyeret, ha mindegyik csak egyetlentől rendelhet!

E feladatok közös jellemzője, hogy eredményük egy sorozat. E sorozat minden egyes tagját valamilyen sorozatból kell kikeresni, de az egyes keresések összefüggenek egymással (vezért nem lehet oda tenni, ahol egy korábban letett vezér ütné; egy munkát nem lehet két munkásnak adni; ha egy pékségnek elfogyott a kenyere, akkor attól már nem lehet rendelni).

Bemenet : $N \in \mathbb{N}_0$, $M \in \mathbb{N}^N$, $\forall i (1 \leq i \leq N): S_i \in H^{M_i}$,
 $fk: N \times H \times H^* \rightarrow L$, $ft: N \times H \rightarrow L$

Kimenet : $VAN \in L$, $X \in \mathbb{N}^N$

ef : $\forall X \in \mathbb{N}^N: fk(i, S_i(X_i); S_1(X_1) \dots S_{i-1}(X_{i-1})) \Rightarrow$
 $\forall j < i: fk(j, S_j(X_j); \dots)$

uf : $VAN \equiv \exists Y \in \mathbb{N}^N: \text{Megoldás}(Y) \text{ és } VAN \Rightarrow \text{Megoldás}(X)$

Definíció: $\text{Megoldás}(X) = (\forall i (1 \leq i \leq N): 1 \leq X_i \leq M_i \text{ és } \forall i (1 \leq i \leq N):$
 $fk(i, S_i(X_i), S_1(X_1), \dots, S_{i-1}(X_{i-1})) \text{ és } ft(i, S_i(X_i)))^1$

Megállapíthatjuk tehát, hogy minden egyes új választás az összes korábbitól függhet (ezt formalizálja az fk függvény), a későbbiekétől azonban nem! Egyes esetekben nemcsak a korábbiaktól, hanem saját jellemzőjétől is függhet a választás (ezt írja le az ft függvény).

Egyes speciális esetekben a korábbiaktól függés a korábbi elemekre külön-külön megvizsgálható (F37, F38), ekkor az előfeltétel triviálisan igaz:

Bemenet : $N \in \mathbb{N}_0$, $M \in \mathbb{N}^N$, $\forall i (1 \leq i \leq N): S_i \in H^{M_i}$,
 $fk: N \times H \times N \times H \rightarrow L$, $ft: N \times H \rightarrow L$

Kimenet : $VAN \in L$, $X \in \mathbb{N}^N$

ef : -

¹ Szavakkal megfogalmazva: az X akkor megoldás, ha minden X_i önmagában helyes, valamint az összes öt megelőző X -beli elem szerint is helyes.

uf : $VAN \equiv \exists Y \in \mathbb{N}^N: \text{Megoldás}(Y) \text{ és } VAN \Rightarrow \text{Megoldás}(X)$

Definíció: $\text{Megoldás}(X) = (\forall i (1 \leq i \leq N): 1 \leq X_i \leq M_i \text{ és}$
 $\forall i, j (1 \leq j < i \leq N): fk(i, S_i(X_i), j, S_j(X_j)) \text{ és } ft(i, S_i(X_i)))$

Változók:

N : **Egész** [a feldolgozandó sorozat száma]
 M : **Tömb** ($1..N$:**Egész**) [a feldolgozandó sorozatok elemszáma]
 VAN : **Logikai** [van-e megfelelő elemsorozat]
 X : **Tömb** ($1..N$:**Egész**) [a megfelelő elemek sorszáma]

A megoldás legfelső szintjén keressünk az I . sorozatból megfelelő elemet! Ha ez sikerült, akkor lépünk tovább az $I+1$. sorozatra, ha pedig nem sikerült, akkor lépünk vissza az $I-1$ -re, s keressünk abban újabb lehetséges elemet!

A keresés befejeződik, ha mindegyik sorozatból sikerült elemet választanunk ($I > N$) vagy pedig a visszalépések miatt már az elsőből sem tudunk újabbat választani ($I < 1$).

Keresés (N, M, X, VAN):

$I := 1$; $X(1..N) := (0, \dots, 0)$

Ciklus amíg $I \geq 1$ és $I \leq N$

Jó_eset_keresés ($M, X, I, melyik, VAN$)

Ha VAN **akkor** $X(I) := melyik$; $I := I + 1$ [előrelép]

különben $X(I) := 0$; $I := I - 1$ [visszalép]

Ciklus vége

$VAN := (I \geq 1)$

Eljárás vége.

Egy lineáris keresés kezdődik az I . sorozatban:

Jó_eset_keresés ($M, X, I, mely, VAN$):

$mely := X(I) + 1$

Ciklus amíg $mely \leq M(I)$ és

$(\text{Rossz_eset}(I, X, mely) \text{ vagy nem } ft(I, mely))^2$

$mely := mely + 1$

Ciklus vége

$VAN := (mely \leq M(I))$

Eljárás vége.

A Rossz_eset függvény az általános esetben a feladattól függő fk függvény kiszámolását jelenti, a fenti speciális esetben pedig egy eldöntést. Ez utóbbit nézzük meg!

Rossz_eset ($I, X, mely$):

$J := 1$

Ciklus amíg $J < I$ és $fk(I, mely, J, X(J))$

$J := J + 1$

Ciklus vége

$\text{Rossz_eset} := (J < I)$

Eljárás vége.

² A specifikációtól eltérünk! Az ft paramétereként az $S(mely)$ az algoritmus végrehajtása közben általában semmit sem mond, ezért nyugodtan helyettesíthetjük magával a 'mely'-lyel.

Az alábbiakban meggondoljuk, hogy miként lehet a fenti keresés tételpárjainak (eldöntés, kiválasztás stb.) backtrack-es változatait megkonstruálni. A fenti megoldást könnyen átalakíthatjuk *eldöntéssé, kiválasztássá, megszámlálássá*, valamint *kiválogatássá*. Az első kettő a legfelső szint minimális átalakítását igényli, a megszámlálást a kiválogatás felhasználja, így ez utóbbit fogalmazzuk meg.

Bemenet : $N \in \mathbf{N}_0$, $M \in \mathbf{N}^N$, $\forall i (1 \leq i \leq N): S_i \in H^{M_i}$,
 $fk: N \times H \times N \times H \rightarrow L$, $ft: N \times H \rightarrow L$
 Kimenet : $DB \in \mathbf{N}$, $Y \in (\mathbf{N}^N)^*$
 ef : -
 uf : $DB = a$ megoldások számának szokásos megfogalmazása
 és $\forall j (1 \leq j \leq DB): \text{Megoldás}(Y_j)$ és Y halmazfelsorolás

A kiválogatás alap gondolata: ha egy megoldást megtaláltunk, akkor azt írjuk ki, majd lépünk vissza, mintha ez nem is lenne megoldás.

Kiválogatás (N, M, DB, Y):
 $I := 1: X(1..N) := (0, \dots, 0): DB := 0$
Ciklus amíg $I \geq 1$
 Jó_eset_keresés($M, X, I, melyik, VAN$)
 Ha VAN **akkor** $X(I) := melyik: I := I+1$ [előrelép]
 különben $X(I) := 0: I := I-1$ [visszalép]
 Ha $I > N$ **akkor** $DB := DB+1: Y(DB) := X: I := I-1$
Ciklus vége
Eljárás vége.

A legtöbb esetben nagyon sok megoldást kapunk, így a kiválogatást általában nem kigyűjtéssel, hanem kiírással végezzük. Van azonban egy eset, amelyben tartósabban szükség lehet az eredményekre: ez a *minimumkeresés*.

Ebben a feladatban a valamilyen szempontból legjobb megoldást kell megadnunk. Megadunk egy kt költségfüggvényt, ami alapján az egyes megoldásokat összehasonlíthatjuk.

Bemenet : $N \in \mathbf{N}_0$, $M \in \mathbf{N}^N$, $\forall i (1 \leq i \leq N): S_i \in H^{M_i}$,
 $fk: N \times H \times N \times H \rightarrow L$, $ft: N \times H \rightarrow L$, $kt: H^+ \rightarrow \mathbf{N}$
 Kimenet : $VAN \in L$, $Y \in (\mathbf{N}^N)^*$
 ef : -
 uf : $VAN = \exists Y \in \mathbf{N}^N: \text{Megoldás}(Y)$ és $VAN \Rightarrow$
 ($\text{Megoldás}(X)$ és $\forall Y: \text{Megoldás}(Y) \Rightarrow kt(Y) \geq kt(X)$)

Az algoritmusban a kiválogatásra alapozva állítsunk elő egy megoldást, ezt hasonlítsuk össze az eddigi legjobb megoldással, s ha jobb nála, akkor ezt őrizzük meg! Kiindu-

ló pontnak vegyünk egy fiktív kezdőértéket, amely minden lehetséges megoldásnál biztosan rosszabb!

Minimumkeresés (N, M, VAN, Y):
 $I := 1: X(1..N) := (0, \dots, 0): DB := 0$
 $VAN := hamis: E := +\infty$ [fiktív érték]
Ciklus amíg $I \geq 1$
 Jó_eset_keresés($M, X, I, melyik, VANJO$)
 Ha $VANJO$ **akkor** $X(I) := melyik: I := I+1$ [előrelép]
 különben $X(I) := 0: I := I-1$ [visszalép]
 Ha $I > N$ **akkor** $VAN := igaz: I := I-1$
 Ha $kt(X) < E$ **akkor** $Y := X: E := kt(X)$
Ciklus vége
Eljárás vége.

A visszalépéses minimumkeresés hatékonyabb lehet abban a speciális esetben, ha a költségfüggvényt az egyes elemekre vett elemibb értékfüggvényei összegeként (feltéve, hogy ezek nemnegatív értékek) kell kiszámolni, azaz ha:

$$kt(X) = \sum_{i=1}^N ertelem(X_i)$$

Ekkor ugyanis a teljes megoldás elkészülte előtt is felfedezhetjük, hogy az éppen készülő megoldás lehet-e még jobb az eddigi legjobbnál.

Minimumkeresés (N, M, VAN, Y):
 $I := 1: X(1..N) := (0, \dots, 0): DB := 0$
 $VAN := hamis: E := +\infty$ [fiktív érték]
Ciklus amíg $I \geq 1$
 Jó_eset_keresés($M, X, I, melyik, VANJO$)
 Ha $VANJO$ **és** $kt(X) + ertelem(melyik) < E$
 akkor $X(I) := melyik: I := I+1$
 különben $X(I) := 0: I := I-1$
 Ha $I > N$ **akkor** $VAN := igaz: I := I-1: Y := X: E := kt(X)$
Ciklus vége
Eljárás vége.

A visszalépéses keresés egy másik változatában nem rögzített elemszámú az eredmény. Egy lehetséges esetben meg kell adni minden olyan (korlátozott elemszámú) sorozatot, amelyet egy nem megfelelő elem zár le.

Definíció: $\text{Megoldás}(X) = \exists K (1 \leq K \leq N): (\forall i (1 \leq i \leq K): 1 \leq X_i \leq M_i \text{ és } \forall i, j (1 \leq j < i \leq N): fk(i, S_i(X_i), j, S_j(X_j)) \text{ és } ft(i, S_i(X_i)) \text{ és } (\exists j (1 \leq j < K \leq N): fk(K, S_K(X_K), j, S_j(X_j)) \text{ vagy } ft(K, S_K(X_K))))$

Alakítsuk át a korábban szerepelt kiválogatási algoritmust úgy, hogy a belső ciklus álljon le, ha vissza kellene lépni! Ekkor jegyezzük fel az aktuális állást, lépünk vissza, majd folytassuk a keresést!

```

Kiválogatás (N,M,DB,Y) :
  I:=1: X(1..N):=(0,...,0): DB:=0
  Ciklus amíg I≥1
    VANJO:=igaz
    Ciklus amíg I≥1 és I≤N és VANJO
      Jó_eset_keresés (M,Y,I,melyik,VANJO)
      X(I):=melyik: I:=I+1
    Ciklus vége
    Ha nem VANJO akkor DB:=DB+1: Y(DB):=X: I:=I-1
  Ciklus vége
Eljárás vége.

```

Itt is megvizsgálhatnánk a legjobb megoldás algoritmusát, de ezt az előző minimum-kereséshez való hasonlósága miatt az Olvasóra hagyjuk.

Egy másik lehetséges esetben nem tudjuk, hogy az eredmény hány elemű, csupán azt, hogy mikor van készen. Ezt a kész (X) logikai értékű függvény adja meg.

```

Kiválogatás (N,M,DB,Y) :
  I:=1: X(1..N):=(0,...,0): DB:=0
  Ciklus amíg I≥1 és nem kész(X)
    Jó_eset_keresés (M,X,I,melyik,VAN)
    Ha VAN akkor X(I):=melyik: I:=I+1 [előrelép]
    különben X(I):=0: I:=I-1 [visszalép]
  Ciklus vége
Eljárás vége.

```

Nézzük meg a fejezet elején közölt feladatok megoldását!

F37. 8 vezér a sakktáblán

Oszloponként egy-egy vezért helyezünk el, a sakktáblán 8 oszlop, oszloponként 8 sor van. Bármelyik két vezér akkor van helyesen elhelyezve, ha nincsenek azonos sorban vagy azonos átlóban.

N=8	a sorozatok száma,
M(i)=N	mindegyik sorozat elemszáma egyforma
ft(a,b) = igaz	azonosan igaz függvény ³
fk(i,j,k,l) = nem (j=l vagy i-k = j-l)	

A keresés legfelső szintje változatlan, a két további eljárást kell újrafogalmazni.

```

Jó_eset_keresés (M,X,I,mely,VAN) :
  mely:=X(I)+1
  Ciklus amíg mely≤M(I) és Rossz_eset(I,X,mely)
    mely:=mely+1
  Ciklus vége
  VAN:={mely≤M(I)}
Eljárás vége.

Rossz_eset (I,X,mely) :
  J:=1
  Ciklus amíg J<I és nem (mely=X(J) vagy I-J=|mely-X(J)|)4
    J:=J+1
  Ciklus vége
  Rossz_eset:=(J<I)
Eljárás vége.

```

F38. N ember - N munka

A megoldást két változatban készítjük el. Az elsőben az egyes emberek által végezhető munkákat egy-egy sorozatban tároljuk.

N	az emberek és az állások száma
M(i)	az i. ember által végezhető munkák száma
S(i,j)	az i. ember által végezhető j. munka
ft(i,j)=igaz	azonosan igaz függvény
fk(i,j,k,l)=(S(i,j)≠S(k,l))	nem végezhetnek azonos munkát

Itt is a két belső eljárást kell újrafogalmazni.

```

Jó_eset_keresés (M,X,I,mely,VAN) :
  mely:=X(I)+1
  Ciklus amíg mely≤M(I) és Rossz_eset(I,X,mely)
    mely:=mely+1
  Ciklus vége
  VAN:={mely≤M(I)}
Eljárás vége.

Rossz_eset (I,X,mely) :
  J:=1
  Ciklus amíg J<I és S(J,X(J))≠S(I,mely)
    J:=J+1
  Ciklus vége
  Rossz_eset:=(J<I)
Eljárás vége.

```

³ A vezér önmagában bárhova tehető.

⁴ I biztosan nagyobb, mint J, ezért nem kell |I-J|-nél az abszolút érték.

A második változatban az emberek tudását egy mátrixban tároljuk, amelyben egy ember és egy munka sorszámahoz rendelünk igaz vagy hamis értéket aszerint, hogy az ember azt a munkát el tudja-e végezni vagy sem.

N	az emberek és az állások száma
N	az i. ember által végezhető munkák száma
S(i,j)	az i. ember el tudja-e végezni a j. munkát
ft(i,j)=S(i,j)	
fk(i,j,k,l)=(j≠l)	nem végezhetnek azonos munkát

Megint a két belső eljárást fogalmazzuk újra.

```

Jó_eset_keresés (M,X,I,mely,VAN) :
  mely:=X(I)+1
  Ciklus amíg mely≤M(I) és (Rossz_eset(I,X,mely) vagy
                                nem S(I,mely))
    mely:=mely+1
  Ciklus vége
  VAN:=(mely≤M(I))
Eljárás vége.

Rossz_eset (I,X,mely) :
  J:=1
  Ciklus amíg J<I és X(J)≠mely
    J:=J+1
  Ciklus vége
  Rossz_eset:=(J<I)
Eljárás vége.

```

F39. N közért - M pékség

Tároljuk a közérték kenyérigényét, a pékségek sütési kapacitását, a közérték és a pékségek kapcsolatát!

N,M	közérték száma, pékségek száma
IGENY(i)	az i. közért kenyérigénye
KAPACITAS(j)	a j. pékség sütési kapacitása
KAPCS(i,j)	van-e kapcsolat az i. közért és a j. pékség között
ft(i,j)=KAPCS(i,j)	

$$fk(i, X(i), X(1), \dots, X(i-1)) = \left(\sum_{\substack{j=1 \\ X(j)=X(i)}}^i IGENY(j) \right) \leq KAPACITAS(X(i))$$

Újra a két belső eljárást fogalmazzuk meg:

```

Jó_eset_keresés (M,X,I,mely,VAN) :
  mely:=X(I)+1
  Ciklus amíg mely≤M(I) és (Rossz_eset(I,X,mely) vagy
                                nem KAPCS(I,mely))
    mely:=mely+1
  Ciklus vége
  VAN:=(mely≤M(I))
Eljárás vége.

Rossz_eset (I,X,mely) :
  S:=0
  Ciklus J=1-től I-ig
    Ha X(I)=X(J) akkor S:=S+IGENY(J)
  Ciklus vége
  Rossz_eset:=(S>KAPACITAS(melyik))
Eljárás vége.

```

Elképzeltető, hogy e feladatnak nincs megoldása. Ekkor módosítsuk úgy a feladatot, hogy olyan megoldást adjunk, amelyben a lehető legtöbb közért kap kenyeret valamilyen pékségtől.

Ezt a -hiányos- megoldást úgy állítjuk elő, hogy felvesszünk egy M+1., fiktív pékséget, amely tetszőleges mennyiségű kenyeret képes sütni, s akivel mindenki kapcsolatban van. Előállítjuk az összes megoldást, s azt választjuk ki közülük, amelyben a legtöbb olyan közért van, amely nem a fiktív szállítótól kapja a kenyeret. A költségfüggvényt kell definiálnunk:

$$kt(X) = \sum_{i=1}^N \chi(X_i \neq M+1), \text{ ahol } \chi(\log) = \begin{cases} 1 & \text{ha } \log \\ 0 & \text{ha } \neg \log \end{cases}$$

A fejezet befejezéseként egy másik "módszert" is leírunk, amellyel némileg egyszerűbb megoldásokat lehet kreálni a backtrack-es tételvariánsokhoz. Egy apró észrevenni valóra építünk, amely után teljesen mechanikusan kaphatók meg az algoritmusok a már jól ismert *alapvariánsaiból*.

Induljunk ki a backtrack első algoritmusának legfelsőbb szintjéből, és hasonlítsuk össze a lineáris keresés tétel algoritmusával. Alighanem a következő analógiákat fedezük föl:

Backtrack-es keresés

```

I:=1 [komponens sorszám]
X(1..N):=(0,...,0) [kezdőindexek]
I≥1 [van-e még keresnivaló]
I≤N [van-e még megoldáskomponens]
Jó_eset_keresés(I,mely,VAN)
Ha VAN akkor X(I):=mely: I:=I+1
különben X(I):=0: I:=I-1
[ a következő komponens ]

```

Nézzük ez alapján pl. az *kiválasztást*!

Kiválasztás (N,M,X) :

```

I:=1: X(1..N):=(0,...,0)
Ciklus amíg I≤N
Jó_eset_keresés(M,X,I,mely,VAN)
Ha VAN akkor X(I):=mely: I:=I+1
különben X(I):=0: I:=I-1
Ciklus vége
Eljárás vége.

```

A *megszámolás* és a *kiválogatás* tételek származtatásának egy látszólagos akadály van. Nevezetesen az, hogy abban nem 'amíg'-os ciklus szerepel. Ezen azonban keresztül tudunk siklani azzal, ha átírjuk a számlálós ciklust amíg-osra:

Ciklus I:=1-től N-ig	I:=1
...	Ciklus amíg I≤N
...	...
Ciklus vége	I:=I+1
	Ciklus vége

Az átírásnak most már semmi akadály nincs. Pl. a *megszámolás* az alábbi:

Megszámolás (N,M,X,DB) :

```

DB:=0
I:=1: X(1..N):=(0,...,0)
Ciklus amíg I≥1
Ha nem I≤N akkor
DB:=DB+1
X(I):=0: I:=I-1
különben
Jó_eset_keresés(M,X,I,mely,VAN)
Ha VAN akkor X(I):=mely: I:=I+1
különben X(I):=0: I:=I-1
Elágazás vége
Ciklus vége
Eljárás vége.

```

Lineáris keresés

```

I:=1 [elemsorszám]
I≤N [van-e még elem]
T(X(I)) [T tul-e]
I:=I+1 [köv. elem]

```

Kiválasztás (N,M,X) :

```

I:=1
Ciklus amíg nem T(X(I))
I:=I+1
Ciklus vége
... Eljárás vége.

```

V. Programozási tételek egymásraépítése

Gyakran előfordul, hogy programozási tételeket egymás után kell használnunk. Ezen egymásutániságnál azonban sok esetben a két megoldóalgoritmus egybeépítése egyszerűbb, rövidebb, hatékonyabb megoldást eredményez. Ebben a részben ezekkel foglalkozunk. Az egymásraépítés mindig két programozási tétel összefogását jelenti, a fejezeteket a korábban alkalmazandó tétel szerint fogalmaztuk meg.

1. Másolással összeépítés

Másolással bármelyik programozási tétel egybeépíthető, hiszen csupán annyi a teendő, hogy a programozási tételben szereplő X_i bemenő adatra hivatkozást kicseréljük $g(X_i)$ -re.

Ha például egy számsorozat elemei négyzetösszegét kellene megadnunk, az egy *másolást* (számokhoz számok négyzetei rendelése) és egy *sorozatszámítást* (összegzés) tartalmaz. Nézzük meg e két tétel általános egymásraépítését!

<i>Bemenet</i> :	$N \in \mathbb{N}_0, X \in H^N, F: G^* \rightarrow G, g: H \rightarrow G^1$
<i>Kimenet</i> :	$S \in G$
<i>ef</i> :	$\exists F_0 \in G \text{ és } \exists f: G \times G \rightarrow G \text{ és}$ $F(Y_1, \dots, Y_N) = f(F(Y_1, \dots, Y_{N-1}), Y_N) \text{ és } F() = F_0$
<i>uf</i> :	$S = F(g(X_1), \dots, g(X_N))$

Az algoritmus:

Függvény

$g: H_{\text{elemtípus}} \rightarrow G_{\text{elemtípus}}$

Változók:

N : **Egész** [a feldolgozandó sorozat elemszáma]
X : **Tömb**(1..N:H_elemtípus) [feldolgozandó elemek]
F0: G_Elemtípus [a művelet nulleleme]
S : G_Elemtípus [az eredmény]

A megoldásban –mint azt a sorozatszámításnál tettük– induljunk ki a nullelemből, alkalmazzuk a f függvényt az addig kiszámított értékre és a sorozat eleméből kiszámított értékre!

¹Az algoritmus szempontjából adottságnak, másként bemenetnek tekinthetjük az F és a g függvényeket is.

Másolás_sorozatszámítás (N, X, S) :

S:=F0
Ciklus I=1-től N-ig
 S:=f(S, g(X(I)))
Ciklus vége
Eljárás vége.

Második példaként vegyük a *másolás* és a *maximumkiválasztás* összeépítését! Ebben a maximális elem értékét és az indexét is meghatározzuk. (Az előző feladat analógiájára ilyen lehet a legnagyobb abszolútértékű szám abszolútértékének meghatározása egy számsorozatból.)

Bemenet : $N \in \mathbb{N}_0$, $X \in H^N$, G rendezett halmaz ($<, \leq$ relációk),
 $g: H \rightarrow G$
 Kimenet : $MAX \in \mathbb{N}$
 ef : $N \geq 1$
 uf : $1 \leq MAX \leq N$ és $\forall i (1 \leq i \leq N) : g(X_{MAX}) \geq g(X_i)$

Az algoritmus:**Függvény**

g: H_elemtípus \rightarrow G_elemtípus

Változók:

N : **Egész** [a feldolgozandó sorozat elemei száma]
 X : **Tömb**(1..N:H_Elemtípus) [a feldolgozandó sorozat elemei]
 MAX: **Egész** [a maximális értékű elem sorszáma]
 MAXERT: G_elemtípus [a maximális érték]

Ebben a megoldásban –a maximumkiválasztás alapján– vegyük az első elemből kiszámított függvényértéket induló maximumnak, ezt hasonlítsuk a további elemekből kiszámított függvényértékekkel, s a legnagyobbat őrizzük meg!

Másolás_maximumkiválasztás (N, X, MAX, MAXERT) :

MAX:=1: MAXERT:=g(X(1))
Ciklus I=2-től N-ig
 Ha MAXERT < g(X(I)) akkor MAXERT:=g(X(I)): MAX:=I
Ciklus vége
Eljárás vége.

2. Megszámolással összeépítés

A megszámlálást három elemi programozási tétellel érdemes egybeépíteni, az *eldöntéssel*, a *kiválasztással*, valamint a *kereséssel*.

Itt olyan kérdéseket tehetünk fel, hogy van-e egy sorozatban legalább K db T tulajdonságú elem, adjuk meg a sorozat K. T tulajdonságú elemét stb.

Az általánossága miatt nézzük a *megszámolás* és a *keresés* egymásraépítését! Az *eldöntésnél*, illetve a *kiválasztásnál* hasonlóan kellene eljárunk, hiszen e két típusalgoritmus megoldásszövege része a *keresés* megoldásszövegének.

Bemenet : $N \in \mathbb{N}_0$, $X \in H^N$
 $T: H \rightarrow \mathbb{L}$, $\chi: H \rightarrow \{0, 1\}$, $\chi(x) = \begin{cases} 1, & \text{ha } T(x) \\ 0, & \text{egyébként} \end{cases}$
 Kimenet : $VAN \in \mathbb{L}$, $SORSZ \in \mathbb{N}$
 ef : –
 uf : $DB = \sum_{i=1}^N \chi(X_i)$ és $VAN \equiv (DB \geq K)$ és
 $VAN \Rightarrow 1 \leq SORSZ \leq N$ és $T(X_{SORSZ})$ és $\sum_{i=1}^{SORSZ} \chi(X_i) = K$

Az algoritmus:**Függvény:**

T: Elemtípus \rightarrow Logikai

Változók:

N : **Egész** [a feldolgozandó sorozat elemei száma]
 X : **Tömb**(1..N:Elemtípus) [a feldolgozandó sorozat elemei]
 K : **Egész** [az eredmény - a megfelelő elemek száma]
 VAN: **Logikai** [az eredmény - van-e megfelelő elem]
 SORSZ: **Egész** [az eredmény - a megfelelő elem sorszáma]

Induljunk ki a keresés megoldásából! A keresés ciklusa akkor állt le, amikor megtaláltuk az első T tulajdonságú elemet. Ezt kell kicserélni arra, hogy csak a K.-nál álljon le (ha egyáltalán van K.). A ciklusmagban viszont számolnunk kell a T tulajdonságú elemeket - ahogyan azt a *megszámolásnál* tettük!

Keresés (N, X, K, VAN, SORSZ) :

I:=0: DB:=0
Ciklus amíg I < N és DB < K
 I:=I+1
 Ha T(X(I)) akkor DB:=DB+1
Ciklus vége
 VAN:=(DB=K)
 Ha VAN akkor SORSZ:=I
Eljárás vége.

3. Maximumkiválasztással összeépítés

Maximumkiválasztással kapcsolatban azt a kérdést fogalmazhatjuk meg, hogy hány darab maximális értékű elem van, s hogy melyek a maximális értékű elemek.

Itt tehát a *maximumkiválasztást* kell egybeépíteni a *megszámolással*, illetve a *kiválogatással*.

Az a kérdés, hogy van-e egyáltalán több maximális értékű elem, egy menetben nem dönthető el, azaz a három tételt nem lehet egymásbaépíteni, hanem csak egymás után alkalmazni.

A korábbiakban megállapítottuk, hogy a kigyűjtéses *kiválogatás* mindig tartalmaz egy *megszámolást*, így csak a *kiválogatással* kell foglalkoznunk.

Bemenet : $N \in \mathbb{N}_0$, $X \in H^N$, H rendezett halmaz
 $\chi: H \rightarrow \{0,1\}$, $\chi(x) = \begin{cases} 1, & \text{ha } x \geq \max(X) \\ 0, & \text{egyébként} \end{cases}$
 Kimenet : $DB \in \mathbb{N}$, $Y \in \mathbb{N}^N$, $\text{MAXERT} \in H$
 ef : -
 uf : $DB = \sum_{i=1}^N \chi(X_i)$ és $Y \in \{1..N\}^{DB}$ és Y halmazfelsorolás
 és $\forall i, j (1 \leq i \leq N, 1 \leq j \leq DB): X_i \leq X(Y_j)$ és $\text{MAXERT} = X(Y_1)$

Az algoritmus:

Változók:

N : **Egész** [a feldolgozandó sorozat elemei száma]
 X : **Tömb**($1..N$:Elemtípus) [a feldolgozandó sorozat elemei]
 DB : **Egész** [a maximális elemek száma]
 Y : **Tömb**($1..N$:Egész) [a maximális elemek sorszámai]

Az összeépítés alap gondolata, hogy a lokális maximumok megőrzésével együtt változassuk is ki a lokális maximumokat. Természetesen új lokális maximum megtalálásakor a korábbi kigyűjtést el kell felejteni. A kiválasztó ciklus végén a lokális maximum éppen a keresett maximumérték, tehát az éppen kigyűjtött sorszárok a maximumok sorszámai lesznek.

Maximumkiválogatás ($N, X, DB, Y, \text{MAXERT}$) :

$\text{MAXERT} := X(1)$; $DB := 1$; $Y(DB) := 1$

Ciklus $I = 2$ -től N -ig

Elágazás

$X(I) > \text{MAXERT}$ **esetén** $\text{MAXERT} := X(I)$; $DB := 1$; $Y(DB) := I$

$X(I) = \text{MAXERT}$ **esetén** $DB := DB + 1$; $Y(DB) := I$

Elágazás vége

Ciklus vége

Eljárás vége.

4. Kiválogatással összeépítés

Elsőként a *kiválogatást* a *sorozatszámítással* építjük egybe. Olyan feladatokról alkalmazható ez, amikor a számítást egy sorozat T tulajdonságú elemeire kell csak elvégeztünk.

Bemenet : $N \in \mathbb{N}_0$, $X \in H^N$, $F: H^* \rightarrow H$
 $T: H \rightarrow L$, $\chi: H \rightarrow \{0,1\}$, $\chi(x) = \begin{cases} 1, & \text{ha } T(x) \\ 0, & \text{egyébként} \end{cases}$
 Kimenet : $DB \in \mathbb{N}_0$, $Y \in \mathbb{N}^N$, $S \in H$
 ef : $\exists F_0 \in H$ és $\exists f: H \times H \rightarrow H$ és
 $F(X_1, \dots, X_N) = f(F(X_1, \dots, X_{N-1}), X_N)$ és $F() = F_0$
 uf : $DB = \sum_{i=1}^N \chi(X_i)$ és $Y \in \{1..N\}^{DB}$ és Y halmazfelsorolás
 és $\forall i (1 \leq i \leq DB): T(X(Y_i))$ és $S = F(X(Y_1), \dots, X(Y_{DB}))$

Az algoritmus:

Függvény:

T : Elemtípus \rightarrow Logikai

Változók:

N : **Egész** [a feldolgozandó sorozat elemei száma]
 X : **Tömb**($1..N$:Elemtípus) [a feldolgozandó sorozat elemei]
 F_0 : Elemtípus [a művelet nulleleme]
 S : Elemtípus [az eredmény]

A megoldásban vegyük a sorozatszámítás algoritmusát, de a műveletet ne minden esetben végezzük el, hanem csak akkor, ha a megfelelő elem T tulajdonságú! Sokszor nincs az eredményhez szükség a DB -re, a mostani megoldásban ezt is képezzük.

Kiválogatás sorozatszámítás (N, X, S, DB) :

$S := F_0$; $DB := 0$

Ciklus $I = 1$ -től N -ig

Ha $T(X(I))$ **akkor** $DB := DB + 1$; $S := f(S, X(I))$

Ciklus vége

Eljárás vége.

Második példánkban a *kiválogatást* a *maximumkiválasztással* építjük egybe. Tipikusan ilyen feladatok azok, amelyekben meg kell határozni egy sorozat T tulajdonságú elemeinek maximumát, minimumát.

Bemenet : $N \in \mathbb{N}_0$, $X \in H^N$, $VA \in L$
 $T: H \rightarrow L$, $\chi: H \rightarrow \{0,1\}$, $\chi(x) = \begin{cases} 1, & \text{ha } T(x) \\ 0, & \text{egyébként} \end{cases}$
 Kimenet : $MAX \in \mathbb{N}$, $MAXERT \in H$


```

ef      : -
uf      : VAN=( $\exists i (1 \leq i \leq N) : T(X_i)$  és
          VAN  $\Rightarrow 1 \leq MAX \leq N$  és  $T(X_{MAX})$  és
          MAXERT= $X_{MAX}$  és  $\forall i (1 \leq i \leq N) : T(X_i) \Rightarrow X_{MAX} \geq X_i$ 

```

Az algoritmus:

Függvény:

T: Elemtípus \rightarrow Logikai

Változók:

N : **Egész** [a feldolgozandó sorozat elemei száma]
X : **Tömb**(1..N:Elemtípus) [a feldolgozandó sorozat elemei]
VAN: **Logikai** [van-e maximális T tulajdonságú elem]
MAXERT: Elemtípus [a maximális értékű elem értéke]
MAX: **Egész** [a maximális értékű elem sorszáma]

Vegyük a megoldáshoz a maximumkiválasztás algoritmusát! Ne akkor jegyezzük fel az új elemet maximumnak, amikor az nagyobb az addigi maximumnál, hanem ennek még az is legyen a feltétele, hogy az új elem T tulajdonságú! Csupán egyetlen probléma van: semmi nem garantálja, hogy az első elem T tulajdonságú, sőt hogy egyáltalán ilyen elem lesz (amit pedig a maximumkiválasztás használ).

Egyik lehetséges megoldás lenne, ha megkeresnénk az első T tulajdonságú elemet, s innen indítanánk a maximumkiválasztást. A másik –s mi ezt követjük– egy fiktív kezdőértékkel indít, s előlről vizsgálja a sorozatot. (Ha a fiktív érték marad a MAXERT-ben az azt jelenti, hogy nincs T tulajdonságú elem a sorozatban.)

Kiválogatás_maximumkiválasztás (N, X, VAN, MAX, MAXERT) :

MAXERT:= $-\infty$

Ciklus I=1-től N-ig

Ha T(X(I)) és X(I) > MAXERT **akkor** MAXERT:=X(I) : MAX:=I

Ciklus vége

VAN:= (MAXERT $\neq -\infty$)

Eljárás vége.

A harmadik példában a kiválogatást a másolással építjük egybe. Ezt olyan feladatoknál kell alkalmazni, ahol egy sorozat T tulajdonságú elemeit kell lemásolni, rajtuk egy függvény kiszámításával.

Bemenet : $N \in \mathbb{N}_0$, $X \in H^N$

$T: H \rightarrow L$, $f: H \rightarrow G$, $\chi: H \rightarrow \{0, 1\}$, $\chi(x) = \begin{cases} 1, & \text{ha } T(x) \\ 0, & \text{egyébként} \end{cases}$

Kimenet : $DB \in \mathbb{N}_0$, $Y \in N^N$, $Z \in G^N$

```

ef      : -
uf      : DB= $\sum_{i=1}^N \chi(X_i)$  és  $Y \in \{1..N\}^{DB}$  és Y halmazfelsorolás
          és  $\forall i (1 \leq i \leq DB) : T(X(Y_i))$  és  $\forall i (1 \leq i \leq DB) : Z_i = f(X(Y_i))$ 

```

Az algoritmus:

Függvény:

T: H_Elemtípus \rightarrow Logikai

f: H_elemtípus \rightarrow G_elemtípus

Változók:

N : **Egész** [a feldolgozandó sorozat elemei száma]
X : **Tömb**(1..N:H_elemtípus) [feldolgozandó elemek]
DB: **Egész** [a megfelelő elemek száma]
Z : **Tömb**(1..N:G_elemtípus) [feldolgozott elemek]

A megoldásban vegyük a kiválogatás kigyűjtéses algoritmusát, de ne a megfelelő elemek sorszámaikat gyűjtjük ki, hanem az ezen elemeken kiszámított függvényértéket!

Kiválogatás_másolás (N, X, DB, Z) :

DB:=0

Ciklus I=1-től N-ig

Ha T(X(I)) **akkor** DB:=DB+1 : Z(DB):=f(X(I))

Ciklus vége

Eljárás vége.

E fejezet algoritmusai nagyon hasonlóan alkalmazhatók *szérválogatással*, *metsszettel*, *unióval* való egymásraépítésre. A szükséges módosításokat a *kiválogatás* ciklusa helyett a megfelelő programozási tétel ciklusában kell elvégezni. A *szérválogatás* specialitása lehet, hogy az eredményeként szereplő két sorozatra különböző programozási tételeket is lehetne alkalmazni (például lehetne feladat a T tulajdonságú elemek maximumának és a nem T tulajdonságú elemek összegének meghatározása a feladat).

Irodalomjegyzék

1. O.J.Dahl-E.W.Dijkstra-C.A.R.Hoare: Strukturált programozás.
Műszaki Könyvkiadó, 1978
2. Varga L.: Programok analízise és szintézise.
Akadémiai Kiadó, 1981
3. N. Wirth: Algoritmusok + adatstrukturák = programok.
Műszaki Könyvkiadó, 1982
4. Fóthi Á.: Bevezetés a programozáshoz.
Tankönyvkiadó, egyetemi jegyzet, 1983
5. Szlávi P.-Zsakó L.: A programozás ABC-je = az ABC' programozása.
ELTE TTK Informatikai Tanszékcsoport, ABC-s füzetek, 1984
6. Szlávi P.-Zsakó L.: Módszeres programozás.
Műszaki Könyvkiadó, 1986
7. Szlávi P.-Zsakó L. et al.: Számítástechnika középfokon.
OMIKK, 1987
8. D.Knuth: A számítógép programozás művészete 3.
Műszaki Könyvkiadó, 1988
9. T.H.Cormen-C.E.Leiserson-R.L.Rivest: Algoritmusok.
Műszaki Könyvkiadó, 1998
10. Rónyai I.-Ivanyos G.-Szabó R.: Algoritmusok.
TYPOTEX, 1999

A *μ*lógia sorozat eddig megjelent tagjai

1. Horváth László - Szlávi Péter - Zsakó László: Modellezés és szimuláció
2. Szlávi Péter - Zsakó László: Programozási forgácsok - KÖMAL feladatmegoldás²
3. Turcsányiné Szabó Márta: A Commodore-64 Terrapin LOGO leírása
4. Szlávi Péter - Zsakó László: Módszeres programozás: Rekurzio
5. Szlávi Péter: A számítógépről népszerűsítő stílusban
6. Zsakó László: Módszeres programozás: Hatékonyság
7. Makány György: Programozási nyelvek: PROLOGIKA
8. Szlávi Péter: A programkészítés technológiája
9. Szlávi Péter - Zsakó László: Szimulációs modellek a populációbiológiában
10. Pintér László: Programozási tételek rekurzív megvalósítása
11. Temesvári Tibor: Alkalmazói rendszerek: QUATTRO PRO 3.0
12. Szlávi Péter - Zsakó László: Módszeres programozás: Adatfeldolgozás
13. Krammer Gergely: Turbo Grafika
14. Pap Gáborné - Szlávi Péter - Zsakó László: Módszeres programozás: Szövegfeldolgozás
15. Pintácsi Imréné - Siegler Gábor - Zsakó László: Szimulációs modellek a kémiában
16. Horváth László - Szabadhegyi Csaba - Szlávi Péter - Zsakó László: Függvényábrázolás
17. Szabadhegyi Csaba - Szlávi Péter - Zsakó László: Szimulációs modellek a fizikában
18. Szlávi Péter - Zsakó László: Módszeres programozás: Programozási bevezető
19. Szlávi Péter - Zsakó László: Módszeres programozás: Programozási tételek
20. Hack Frigyes: Számítógéppel támogatott problémamegoldás
21. Szlávi Péter - Temesvári Tibor - Zsakó László: Módszeres programozás: A programkészítés technológiája
22. Szlávi Péter - Temesvári Tibor - Zsakó László: Programozási nyelvek: Alapfogalmak
23. Illés Zoltán: Programozási nyelvek: C++
24. Szlávi Péter - Temesvári Tibor - Zsakó László: Programozási nyelvek: ELAN
25. Ökrös László: Programozási nyelvek: Az LCN LOGO leírása
26. Hack Frigyes: Informatika
27. Pap Gáborné - Szlávi Péter - Zsakó László: Módszeres programozás: Rekurzív típusok
28. Hack Frigyes: Programozási nyelvek: BASIC

²A *μ*lógia-sorozat dőlt betűkkel szedett tagjai már nem kaphatók.

29. *Kincses Zoltán: Gólyakalauz az Internet használatához*
30. *Zsakó László: Az informatika ismeretkörei*
31. *Hack Frigyes: Informatikai ismeretek*
32. *Hack Frigyes: DERIVE*
33. *Pozsár Erika: Internet és társadalmi viszonyok*
34. *Pap Gáborné - Szlávi Péter - Zsakó László: Módszeres programozás: Adattípusok*
35. *Köves Gabriella: TEX lépések*
36. *Nagy István: Szövegszerkesztés a Word 97-tel*
37. *Gábor Béla: Programozási nyelvek: A Delphi programozása*
38. *Szlávi Péter - Zsakó László: Módszeres programozás: Gráfok, gráfalgoritmusok*
39. *Fejezetek a számítástechnika történetéből I.*
40. *Fejezetek a számítástechnika történetéből II.*
41. *Vágvölgyi István: Prezentáció és grafika oktatása*
42. *Nagy Sándorné: Adatbázis-kezelés Access 97-tel*
43. *Hack Frigyes: 3D-grafika geometriai alapjai*
44. *Zsakó László (szerk.): Fejezetek a számítógépi grafikából*
45. *Dávid András – Pap Gáborné: Adatszerkezetek példatár*

Szilánkok részsorozat³

1. *Szlávi Péter: A típusfogalom fejlődése az algoritmikus nyelvekben.*
2. *Temesvári Tibor: ELANI programozási nyelv leírása*
3. *Szlávi Péter: Előadás a szövegtípusokról*
4. *Szlávi Péter: Előadás a rekurzióról*
5. *Szlávi Péter: Előadás a gráftípusról*
6. *Szlávi Péter: Adatok, adattípusok*
7. *Szlávi Péter: Előadás a sorozattípusokról*
8. *Szlávi Péter: Előadás a file-típusokról és a táblázattípusról*
9. *Szlávi Péter: Előadás a táblázattípusról*
10. *Gálos György - Pap Gáborné: Adatszerkezetek példatár*
11. *Szlávi Péter: Gondolatok a típus-specifikációk kompatibilitásának vizsgálatáról*

³A Mikrológia sorozat előzeteseként megjelent könyvek, melyek később beépülnek a sorozatba. (A dőlt betűkkel szedettek ilyenek, emiatt már nem kaphatók.)