INFORMATIKA

Sorozatok nevezetes algoritmusai – példatár

Összeállította:
Dr. Baksáné dr. Varga Erika

Gépészmérnöki és Informatikai Kar
Informatikai Intézet

MISKOLCI EGYETEM

2014

TARTALOMJEGYZÉK

1. Bevezetés

A jegyzet szerzője a Miskolci Egyetem Informatikai Intézetének gondozásában lévő, informatikus alapképzésben első félévben oktatott *Programozás alapjai* tantárgy oktatója. A tantárgy címe és a tantervben elfoglalt helye is mutatja, hogy alapozó, bevezető jellegű és az informatikus jelöltek számára alapvető fontosságú témaköröket foglal magába. Fő célja a C programozási nyelv ismertetésén keresztül bemutatni a számítógép felépítését és működését, az adatok tárolásának és feldolgozásának módját. Bármely programozási nyelv tanulását segíti, ha nemcsak a szintaktikai elemek elsajátítására, hanem gyakorlati problémák megoldására is törekszünk. A természetes nyelvek tanulása sem pusztán a szókincs bővítését jelenti; a cél a kommunikációkészség kialakítása, javítása kell legyen. Nincs ez másként a számítógéppel történő kommunikáció esetén sem. Attól hogy megtanuljuk egy programozási nyelv jelkészletét és szabályrendszerét, még nem fogunk tudni használható és hatékony programot írni. Ezért a C programozási nyelv bemutatásán túl általános programozási elveket (tételeket) is oktatunk e tantárgy keretében. A programozási tételek olyan általánosan megfogalmazott, egyszerűbb algoritmusok, amelyek ismerete segítségünkre van összetettebb algoritmizálási feladatok megoldásában: egyrészt mert mintaként szolgálnak, másrészt mert ismeretük segítséget nyújt a probléma részfeladatokra történő bontásában. Ezen alapelvek minden, a számítógép programozásához kapcsolódó tantárgy alapját képezik. Nem csoda hát, hogy ebben a témában igen gazdag szakirodalomból válogathatunk, többek között: [1], [2], [3], [4], [5], [6].
Miért van szükség mégis egy újabb jegyzetre? Azért, mert a szerző tapasztalata szerint az alapképzésben résztvevő hallgatók segítséget igényelnek ahhoz, hogy kiválogassák a rendelkezésre álló információhalmazból a számukra releváns elemeket. Ráadásul ez a jegyzet a sorozatokra vonatkozó nevezetes és alapvető algoritmusok bemutatásán túl megoldott, magyarázattal ellátott példákat és gyakorló feladatokat is tartalmaz. Miután a megoldások nem egy konkrét programozási nyelven adottak, a szerző szándéka szerint a számítógép programozás oktatásában széleskörűen felhasználható, az önálló hallgatói felkészülést segítő elméleti és gyakorlati tananyagot tart kezében az olvasó, amit remélhetőleg minden kezdő és haladó szinten tanuló programozó haszonnal forgat majd a választott programozási nyelvtől függetlenül.

Ajánlott irodalom

Donald E. Knuth: A számítógép-programozás művészete (The Art of Computer Programming), Addison-Wesley Professional

- Első kötet: Alapvető algoritmusok (Fundamental Algorithms), második kiadás, 1994, Budapest, Műszaki Könyvkiadó
- Második kötet: Szeminumerikus algoritmusok (Seminumerical Algorithms), második kiadás, 1994, Budapest, Műszaki Könyvkiadó
- Harmadik kötet: Keresés és rendezés (Keresés és rendezés), második kiadás, 1994, Budapest, Műszaki Könyvkiadó

2. Alapfogalmak

2.1. Adat, adatszerkezet

A számítógép egy adatfeldolgozó eszköz. Az *adat* tények, elemi ismeretek olyan megjelenési formája, amely alkalmas a számítógéppel történő értelmezésre, feldolgozásra, továbbításra. Az adatokból gépi feldolgozás útján információkat, azaz új ismereteket nyerünk. Vagyis az *információ* nem más, mint értelmezett adat.
A számítógép által feldolgozandó adat lehet egyszerű (elemi adat) vagy összetett (adatcsoport); illetve lehet konstans (értéke a feldolgozás során nem változhat meg) vagy változó. Az adatnak mindig van *azonosítója* – az adat neve, amellyel rá, vagyis az értékére hivatkozhatunk – és *típusa*, amely meghatározza, hogy az adat mekkora memóriaterületen, és ezen belül milyen kódolási szabályok szerint tárolódik a számítógépben (ez egyben az ilyen típusú változó által felvehető értékek halmazát is kijelöli). A típus az adat összetettségét is mutatja és behatárolja az adaton elvégezhető műveletek körét. Ezzel a két jellemzővel *deklaráljuk* az adatot. Azáltal, hogy az azonosítóval rendelkező adatot *definiáljuk*, azaz számára memóriát foglalunk a számítógépben, újabb két tulajdonságot rendelünk hozzá: memóriabeli *címet* és *aktuális értéket* (memóriatartalmat). A számítógéppel végrehajtandó feladat *adatstruktúrája* a feladat adatainak összessége, melyek a feladat megoldása során betöltött szerepük alapján lehetnek kiinduló (input) adatok, eredmény (output) adatok vagy munkaadatok.

Az ebben a jegyzetben tárgyalt algoritmusok esetén a kiinduló (bemeneti) adatok nem elemi adatok, hanem adat-sorozatok. Sok esetben egymás után beolvasott elemi adatokról van szó, amelyeket a beolvasást követően azonnal feldolgozunk, vagyis további tárolásukra nincs szükség. Ha azonban el kell tároljuk az adatsorozat elemeit, azt tömbben tesszük. A *tömb* olyan adatcsoport, amelynek elemei azonos típusúak és ún. dimenziók szerint vannak elrendezve. A dimenziószám azt jelenti, hogy hány kijelölő érték (index) kell ahhoz, hogy az adatcsoportból egy elemet kiválasszunk. Az index sorszámjellegű adat. Ilyen értelemben az egydimenziós tömb egy adatsor (vektor, egy indexszel rendelkezik), a kétdimenziós tömb egy téglalap alakú adattáblázat (mátrix, egy sor és egy oszlopindexszel). Ebben a jegyzetben a tömböt úgy jelöljük, hogy az azonosító név után szögletes zárójelben megadjuk a méretét (dimenziónként). A tömb adatcsoport jellemző kezelési módja az elemenkénti feldolgozás, tehát a műveletekben (az I/O műveletekben is) a tömbelemek az operandusok. Ebből következik, hogy az elemek feldolgozása ciklusban történik, ahol felhasználjuk az elemeket kijelölő indexet, amit a tömb neve után szögletes zárójelben adunk meg.

2.2. Algoritmus, algoritmizálás

A számítástechnikában a *program* algoritmusok számítógépes megvalósítását jelenti. Az algoritmus egy bonyolult számítástudományi fogalom. A modern algoritmuselmélet atyjának Alan Mathison Turing (1912-1954) angol matematikust tekintjük, aki az 1930-as években alkotta meg és tette közzé az algoritmus matematikailag pontos fogalmát. Elkészített egy absztrakt (elméleti) gépet, a Turing-gépet, aminek segítségével algoritmuselméleti problémák szimbolikusan kezelhetők, ebben a témakörben tételek mondhatók ki és bizonyíthatók [7].

SZÉCHENYI 2020
MAGYARORSZÁG KORMÁNYA
Európai Unió
Európai Szociális Alap
BEFEKTETÉS A JÖVŐBE

Leegyszerűsítve, az *algoritmus* egy feladat megoldásának véges számú részlépésben történő egyértelmű és teljes leírása. A helyes algoritmussal szemben támasztott legfontosabb követelmények:

- egy értékhez vagy ezek egy halmazához (input) hozzárendel egy értéket vagy ezek egy halmazát (output);
- egyértelmű (az algoritmust alkotó utasítások egyértelmű, meghatározott sorrendben követik egymást);
- determinisztikus (ugyanazon kiindulási adatokra tetszőleges számú végrehajtás esetén ugyanazt az eredményt szolgáltatja);
- véges (véges számú lépés után befejeződik, és eredményre vezet);
- redundáns lépéseket (ismétlődéseket, felesleges utasításokat) nem tartalmaz;
- teljes (nemcsak egy konkrét esetre alkalmazható, hanem az összes azonos jellegű feladatra).

Egy algoritmus elemi (azonnal végrehajtható) és összetett (elemi tevékenységekből felépülő) műveletek sorozata. A *művelet* általános értelemben véve egy olyan átalakítás (transzformáció), amely az adatok aktuális értékeit felhasználva előállítja az adatok új értékeit. Az algoritmus struktúráját a *vezérlő szerkezetek* adják, amelyek tetszőleges mélységben egymásba ágyazhatók, és amelyek a feladat műveletekre bontását, és ezek végrehajtási sorrendjét írják le, beleértve egy-egy művelet adott esetben való elhagyását, vagy éppen többszöri végrehajtását is. Az alapvető vezérlő szerkezetek ismertetésére a 3. szakaszban kerül sor.

Programkészítés során általában kapunk egy feladatot, illetve a feladatot pontosan leíró *specifikációt*. Ez legtöbbször deklaratívan adott. Leírja mit kell csinálnia a programnak, mi a bemenete és a kimenete, és mi ezek között az összefüggés. Nekünk kell kitalálni a hozzá tartozó imperatív megoldást, azaz egy módszert, amellyel a bemenetből a kimenet előállítható. Ez a feladat algoritmusa: a megoldás menete lépésről lépésre, utasítások sorozataként megadva. Az algoritmus tervezésének alapja a dekompozíció. A feladatot részekre kell bontani, a részfeladatokat egyenként meg kell oldani, majd ezeket össze kell állítani, hogy együtt működhessenek. A részekre bontásnál kétféleképpen járhatunk el: vagy alulról felfelé építkezünk (*bottom-up módszer*), ahol a kisebb feladatok megoldásával kezdjük, majd ezeket összeillesztve kapjuk a teljes probléma megoldását; vagy felülről lefelé, fokozatosan, lépésenként finomítjuk az eljárást (*top-down módszer*). Sajnos nem létezik olyan általános módszer, amely segítségével egy algoritmus szisztematikusan kidolgozható lenne, tehát rendszerint nem csak egy megoldás létezik. Minden megoldás elfogadható, amelyre teljesülnek a 2.2. szakaszban megfogalmazott elvárások. Szerencsére a programozás során rengeteg olyan problémával találkozunk, amelyek rendszeresen felmerülnek. Vannak jól bevált megoldások, amelyeket érdemes újra felhasználni, vagyis programozáskor gyakran csak a már megtanult algoritmusokat kódoljuk. Ebben a jegyzetben a sorozatokkal kapcsolatban gyakran előforduló, általános (generikus) algoritmusokat aszerint csoportosítva tárgyaljuk, hogy milyen eredményt (kimenetet) szolgáltatnak.

2.3. Algoritmusok megadása

Az algoritmusok szemléletes, könnyen áttekinthető leírására többféle (főként grafikus) eszköz létezik. Ezek közül ebben a jegyzetben a folyamatábra eszközrendszerét (1. ábra) fogjuk alkalmazni a pszeudokód formális leírás (2. ábra) mellett. A pszeudokód az algoritmusok leírására

használt olyan mesterséges formális nyelv, amely változókból és néhány állandó jelentésű szóból (foglalt konstansból) áll, és (szándékosan) hasonlít a számítógépes programozási nyelvekre.

Algoritmus elejét ill. végét jelző határszimbólumok	
Beolvasó és kiíró utasítás (input/output)	
Elemi utasítás	
Elágazás: a feladat végrehajtása a rombuszba írt feltétel igazságértékétől függően folytatódik	
Csatlakozási pont (elágazás előtt / után)	
A végrehajtási sorrendet kijelölő nyilak	

1. ábra. Folyamatábra elemei

Elemi algoritmus kezdete Elemi algoritmus vége	Eljárás: Eljárás vége
Beolvasó utasítás Kiíró utasítás	Input: *változók* Output: *változók*
Elemi utasítás (értékadás)	változó := kifejezés
Szelekció	HA feltétel AKKOR elemi utasítások EGYÉBKÉNT elemi utasítások FELÉTEL vége
Előltesztelő ciklus	AMÍG feltétel ADDIG elemi utasítások CIKLUS vége
Hátultesztelő ciklus	CIKLUS elemi utasítások AMÍG feltétel

2. ábra. Pszeudokód szerkezetek

2.4. Strukturált programozás

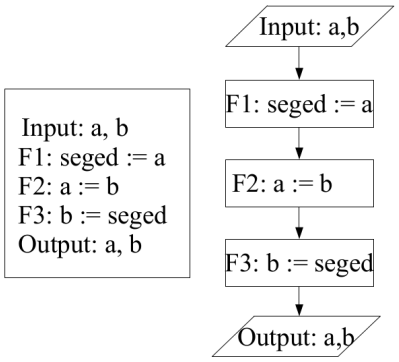
A strukturált programozás egy szabványos (Federal Standard 1037C) programépítési alapelv, ami E.W. Dijkstra nevéhez kötődik [8]. Lényege, hogy a teljes feladat olyan kis feladatelemekre legyen felosztva, amelyek egymással nincsenek átfedésben, egymáshoz meghatározott logika szerint kapcsolódnak, és mindegyik megoldható valamilyen elemi struktúra, elemi programséma követésével. A feladat felosztásakor a részfeladatok csak három szerkezeti mintázat szerint kapcsolódhatnak egymáshoz: *szekvencia*, feltételes elágazás (*szelekció*) vagy feltételes ciklus (*iteráció* vagy véges ismétlés). Az elmélet fontos része az a szabály, hogy ezek az alapelemek egymásba ágyazhatók.

SZÉCHENYI 2020
MAGYARORSZÁG KORMÁNYA
Európai Unió
Európai Szociális Alap
BEFEKTETÉS A JÖVŐBE

3. Alapvető vezérlő szerkezetek

3.1. Szekvencia

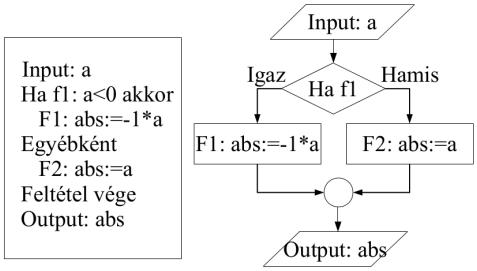
Az F művelet felbontása olyan $F_1, \ldots, F_n$ műveletekre, amelyeknek a felbontás sorrendjében való egymás utáni végrehajtása magának az F-nek a végrehajtását jelenti (3. ábra).



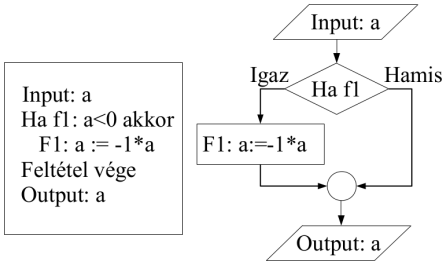
3. ábra. F: két érték felcserélése

3.2. Szelekció

Az F művelet felbontása olyan $F_1, \ldots, F_n$ műveletekre, amelyek közül egy kiválasztása és végrehajtása magának az F műveletnek a végrehajtását jelenti. A kiválasztást egy $f_1, \ldots, f_n$ feltételrendszer szabályozza, ahol az f_i feltétel teljesülése az F_i kiválasztásának a feltétele. A feltételrendszer független feltételekből áll, tehát minden konkrét esetben legfeljebb egy teljesül (4. ábra). Ha egy sem teljesül, akkor a szelekció az üres tevékenységet képviseli (5. ábra).



4. ábra. F: abszolút érték meghatározása



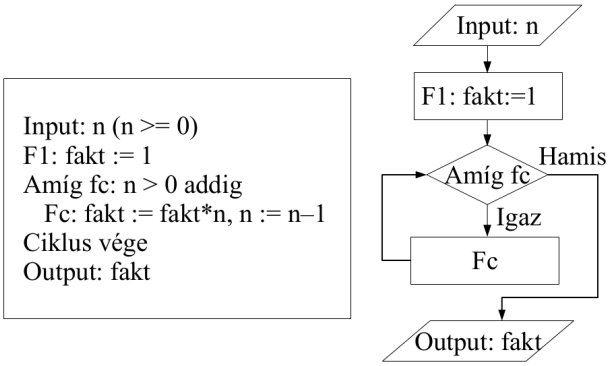
5. ábra. F: abszolút érték meghatározása

3.3. Iteráció

Az F művelet végrehajtása egy F_c műveletnek az egymás utáni ismételt (ciklikus) végrehajtásával. Az ismételtetést egy f_c ciklusfeltétel szabályozza. A ciklusok osztályozása több szempont szerint elvégezhető. Ebben a jegyzetben a feltétel ellenőrzésének módja szerinti két típust tárgyaljuk.

3.3.1. Előltesztelő ciklus

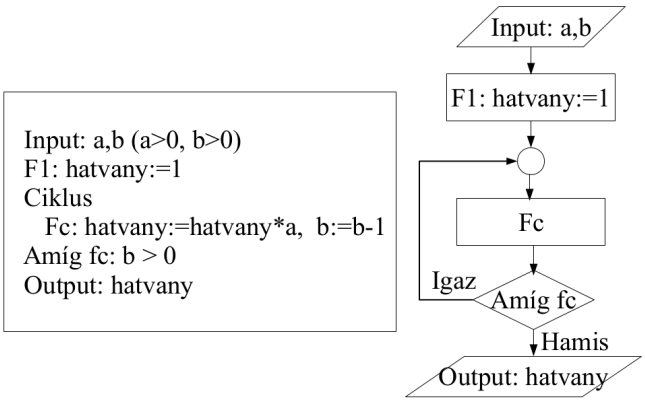
Az f_c feltétel vizsgálata megelőzi az F_c ciklusmag végrehajtását, ami csak akkor következik be, ha a feltétel igaz. Ha a ciklusfeltétel hamis, az F művelet készen van. Ilyen ciklusnál az ismétlések száma lehet 0 is, ez esetben az iteráció az üres tevékenységet képviseli. A 6. ábra példájában feltételezzük, hogy a bemeneti adat pozitív egész szám.



6. ábra. F: faktoriális számítás

3.3.2. Hátultesztelő ciklus

Az F_c ciklusmag egyszeri végrehajtását követi az f_c feltétel vizsgálata, tehát az ismétlések száma legalább 1. Amíg a ciklusfeltétel igaz, a ciklusmag műveletei végrehajtódnak. Az F művelet akkor van kész, amikor a feltétel hamissá válik. A 7. ábrán látható példában feltesszük, hogy a bemeneti adatok pozitív egész értékek.



7. ábra. F: a^b hatványozás

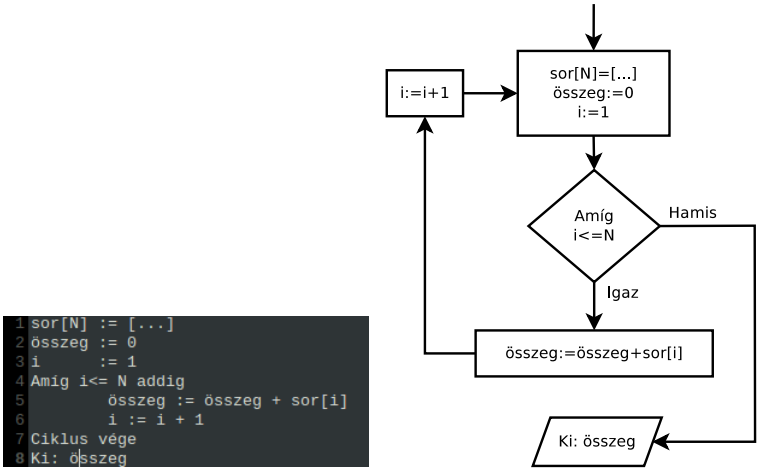
Vegyük észre, hogy az előző két példa mindkét ciklus típussal megoldható. Lényeges eltérés közöttük azonban a ciklusmag első utasítása. A hatványozási feladatban a ciklusmag első utasítása minden lépésben ugyanaz, vagyis nem függ az előző iteráció eredményétől. Ellenben a faktoriális számítás ciklusmagjának első utasításában az egymást követő ismétlések során az

előző ismétlés végeredménye képezi a következő ismétlés kiinduló adatát, ami tipikus példája az iterációs eljárásnak.

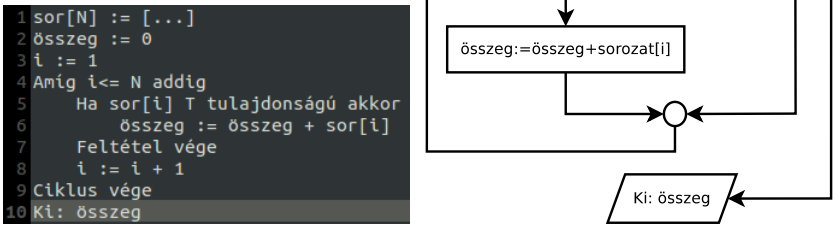
4. Egy sorozathoz egy értéket rendelő algoritmusok

4.1. Összegzés

Adott egy N elemű számsorozat. Feladatunk az összes elem (8. ábra), vagy a T tulajdonsággal rendelkező elemek összegének (9. ábra) kiszámítása.



8. ábra. Számsorozat elemeinek összegzése



9. ábra. Számsorozat T tulajdonságú elemeinek összegzése

A feladat további változataiban az elemek szorzatát, vagy valamilyen átlagát kell meghatározni. Ebbe a feladattípusba tartozik például a faktoriális számítás (6. ábra) és a hatványozás (7. ábra) problémája is.

4.1.1. Példa

Egy valutaváltó hetente könyveli vételi és eladási forgalmát az euró váltása kapcsán. Adjuk meg azt az algoritmust, amellyel a valutaváltó havonta meg tudja határozni a középárfolyamot. **Megoldás:** Egy hónapban 4 hét van, ezért havonta 4 vételi árfolyam, 4 eladási árfolyam, 4 vételi összeg és 4 eladási összeg adat áll rendelkezésünkre a számításhoz. A heti középárfolyam meghatározása az alábbi képlettel történik (harmonikus átlag):

$$(e\_osszeg + v\_osszeg)/(e\_osszeg/e\_arf + v\_osszeg/v\_arf). \tag{4.1}$$

Ebből a havi középárfolyam úgy számítható, hogy ciklusban összegezzük a heti átlagokat majd elosztjuk 4-el. A számításhoz az érthetőség kedvéért két segédváltozót vezetünk be, bár eggyel is megoldható a feladat.

programkód 1. Középárfolyam meghatározása

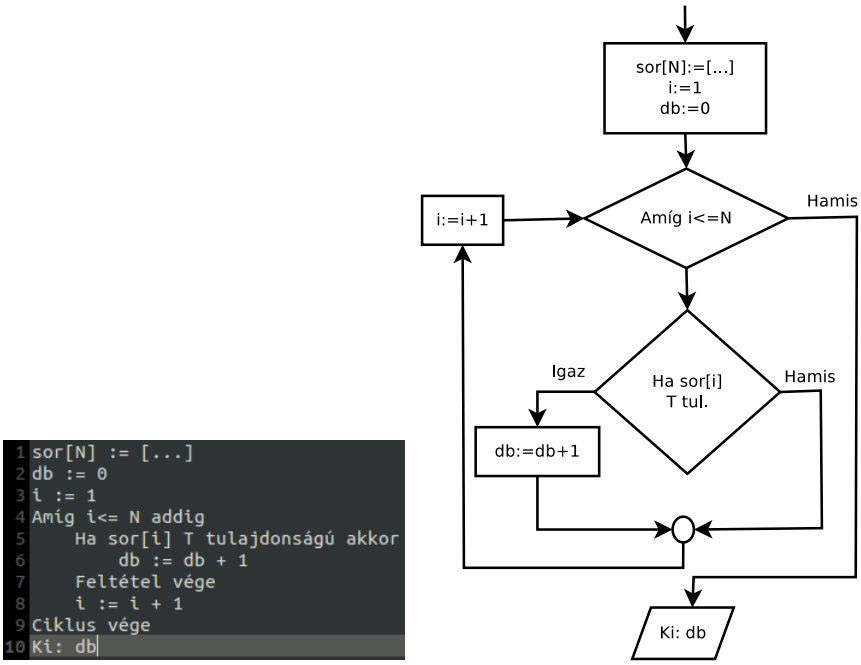
```
Valtozok: osszeg,n,i,v_arf,e_arf,v_osszeg,e_osszeg
i := 1
n := 4
osszeg := 0
Amig i <= n addig
  Input: v_arf, e_arf, v_osszeg, e_osszeg
  kozep_arf := (e_osszeg + v_osszeg)/
            (e_osszeg/e_arf + v_osszeg/v_arf)
  osszeg := osszeg + kozep_arf
  i := i+1
Ciklus vege
Output: osszeg/4
```

4.1.2. Feladatok

1. Határozza meg az első N egész szám mértani átlagát ($N > 0$).
2. Számítsa ki egy számsorozat negatív elemeinek összegét.
3. Számítsa ki egy számsorozat páros elemeinek szorzatát.
4. Állítsa elő a Fibonacci-sorozat n-edik elemét. A sorozat első elemei 0 és 1, majd ezt követően minden szám a megelőző kettő összege.
5. Adott egy tanuló érdemjegyeit tartalmazó számsorozat. Számítsa ki a tanuló tanulmányi átlagát és ösztöndíját az alábbiak szerint: 3,5 alatt 0 Ft; 3,6 – 4,0 között 5e Ft/hó; 4,1 – 4,5 között 10e Ft/hó; 4,6 – 5,0 között 15e Ft/hó.
6. Egy üzletben minden elköltött 2000 ft után 10 ajándékpontot írnak jóvá. Adjuk meg azt az algoritmust, amely a vásárlási összegek ismeretében negyedévente összesíti a gyűjtött pontokat.

4.2. Számlálás

Adott egy N elemű sorozat, és egy, a sorozat elemein értelmezett T tulajdonság. A feladat a T tulajdonsággal rendelkező elemek megszámlálása. A megoldáshoz szükség van egy számláló változóra, amely miközben ciklusban végigvizsgálom az összes elemet, mindig azt mutatja, hogy eddig hány elem tett eleget a feltételnek.



10. ábra. Egy sorozat T tulajdonságú elemeinek megszámlálása

4.2.1. Példa

Számoljuk meg, hogy egy adott n számnak hány osztója van (önmagát is beleértve).

programkód 2. Osztók száma

```
Valtozok: oszto, db, n
Input: n
oszto := n
db := 0
Amig oszto>0 addig
  Ha n mod oszto = 0 akkor
    db := db + 1
  Feltetel vege
  oszto := oszto - 1
Ciklus vege
Output: db
```

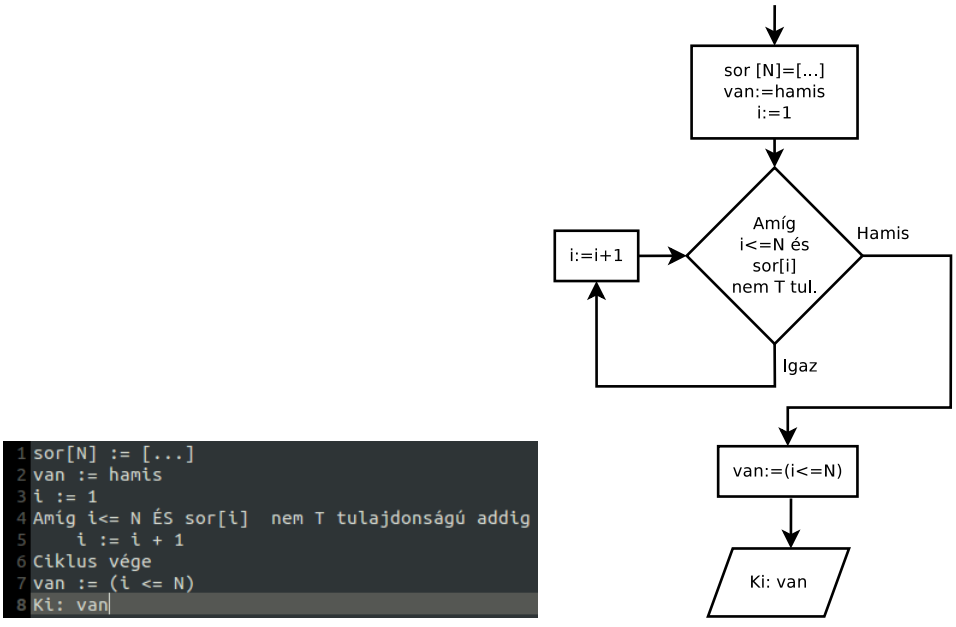
Megoldás: A számláló változót lenullázzuk, és bevezetünk egy osztó változót (amit vagy n-ről indítunk és csökkentünk 1-ig, vagy megfordítva). Ezután ciklusban vizsgájuk az osztókat, hogy osztják-e maradék nélkül n-et. Ha igen, a számlálót növelni kell eggyel.

4.2.2. Feladatok

1. Határozzuk meg N szám közül hány pozitív.
2. Határozzuk meg N szám közül hány páratlan.
3. Egy N karakter hosszúságú szövegben számoljuk meg a magánhangzókat.
4. Júliusban minden nap feljegyeztük a Balaton vízhőmérsékletét. Mondjuk meg hányszor volt hideg a strandoláshoz (20 C fok alatt).
5. Egy oktató zh statisztikát készít, és a 0 pontos dolgozatok százalékos arányára kíváncsi. Adjuk meg a feladat algoritmusát.
6. Egy háziasszony a hét minden munkanapján vásárol. Írjuk meg azt az algoritmust, amellyel a háziasszony hétvégén összesíteni tudja a heti kiadásokat és megszámolja hogy hányszor költött 5000 ft felett.

4.3. Eldöntés

Adott egy N elemű sorozat és egy, a sorozat elemein értelmezett T tulajdonság. Feladatunk annak eldöntése, hogy van-e a sorozatban *legalább egy* T tulajdonsággal rendelkező elem. A megoldás menete a következő. Az algoritmus elején feltesszük, hogy nincs T tulajdonsággal rendelkező elem; majd ciklusban vesszük sorra az elemeket és vizsgáljuk, hogy az adott elem eleget tesz-e a feltételnek. Amíg nem találunk ilyen elemet, keresünk tovább. Ha viszont találunk egy T tulajdonsággal rendelkező elemet, akkor vége az algoritmusnak.




```
1 sor[N] := [...]
2 van := hamis
3 i := 1
4 Amíg i<= N ÉS sor[i]  nem T tulajdonságú addig
5     i := i + 1
6 Ciklus vége
7 van := (i <= N)
8 Ki: van
```

11. ábra. Annak eldöntése, hogy egy sorozatban van-e T tulajdonságú elem

4.3.1. Példa

Egy tanuló érdemjegyei alapján döntsük el, hogy kitűnő tanuló-e (minden jegye 5-ös). A feladat megoldásakor feltesszük, hogy nem kitűnő tanuló; majd ciklusban vizsgáljuk a jegyeit. A ciklus kétféleképpen érhet véget: 1) vagy találunk egy jegyet, amelyik nem ötös (tehát a tanuló nem kitűnő), ekkor $i <= N$ vagyis a kimenet eredménye hamis; 2) vagy a tanuló minden jegye ötös (kitűnő tanuló), így a ciklus végeztével $i > N$ igaz. A bemenet a jegyek sorozata, de mivel a beolvasás után csak egy műveletet végzünk velük, nem szükséges az eltárolásuk.

programkód 3. Kitűnő tanuló-e

```
Valtozok: kituno, i, jegy, n, kilep
Input: n
kituno := hamis
kilep := hamis
i := 1
Amig i <= n ES kilep := hamis addig
  Input: jegy
  Ha jegy=5 akkor
    i := i+1
  Egyebkent
    kilep := igaz
  Feltetel vege
Ciklus vege
Output: kituno := (i>n)
```

4.3.2. Példa

Adott a prímszámok sorozata. Döntsük el, hogy egy adott $n >= 4$ páros szám esetén létezik-e olyan prímpár, amelyre $n = p_1 + p_2$ teljesül [9].

programkód 4. Goldbach-sejtés

```
Valtozok: van, i, primek[N], n
Input: n
van := hamis
i := 1
primek[N] = [ 1, 2, 3, 5, ... ]
Amig i<=N ES van=hamis addig
  j := i
  Amig j<=N ES n!=primek[i]+primek[j] addig
    j := j + 1
  Ciklus vege
  Ha j<=N akkor van := igaz
  i := i + 1
Ciklus vege
Output: van
```

Goldbach sejtése szerint minden $n >= 4$ páros szám esetén létezik legalább egy prímpár ami teljesíti a feltételt. **Megoldás:** Mivel a feltételben két értéket kell vizsgálni, két egymásba ágyazott ciklusra lesz szükségünk a megoldáshoz. Jelen esetben a p_1 értékeket a külső ciklusban az i index, a p_2 értékeket a belső ciklusban a j index jelöli ki. Elvileg mindkét index 1-től N-ig futhatna, de amennyiben a sorozat monoton növekvő gyorsabban eredményre jutunk, ha végiggondoljuk hogy a feltételben az összeadás kommutatív, azaz p_1 és p_2 szerepe felcserélhető, ezért a belső ciklusban elegendő, ha a j index i-től veszi fel az értékeket N-ig. Általánosan megfogalmazva, az alábbi megoldás értékpárokat képez ismétlődés nélkül (ahol (p_1, p_2) és (p_2, p_1) ugyanazt a számpárt jelenti). Az eredmény tárolásához bevezetünk változót, amit kezdetben hamisnak feltételezünk és akkor lesz igaz, ha találunk a feltételt kielégítő számpárt.

Ez az algoritmus nem elég általános, ugyanis a prímszámok sorozatának előállítása nem triviális feladat. Algoritmusunk akkor lesz újra felhasználható, ha tovább bontjuk és mi határozzuk meg egy adott számról, hogy prímszám-e. Módosítsuk tehát úgy a feladatot, hogy legyen adott a számok sorozata N-ig. Egy adott szám prím, ha önmagán és 1-en kívül nincs több osztója.

programkód 5. Goldbach-sejtés általános megoldása

```
  Valtozok: van, i, n, N
  Input: N, n
  van := hamis
  i := 1
  Amig i<=N ES van=hamis ES osztok_szama(i)=0 addig
    j := i
    Amig j<=N ES n!=i+j ES osztok_szama(j)=0 addig
      j := j + 1
    Ciklus vege
    Ha j<=N akkor
      van := igaz
    Feltetel vege
    i := i + 1
  Ciklus vege
  Output: van

Eljaras: osztok_szama
  Valtozok: oszto, db, szam
  Input: szam
  oszto := 2
  db := 0
  Amig oszto < szam/2 addig
    Ha szam mod oszto = 0 akkor
      db := db + 1
    Feltetel vege
    oszto := oszto + 1
  Ciklus vege
  Output: db
Eljaras vege
```

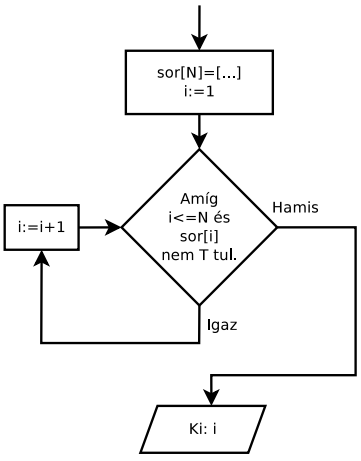
A prímszámok meghatározásához tehát felhasználhatjuk a 2. algoritmust. Megjegyezzük azonban, hogy létezik ennél hatékonyabb (gyorsabban eredményre vezető) megoldás annak megállapítására, hogy egy szám prím-e. Nevezetesen az oszthatóság vizsgálatát 2-vel kezdjük (tudjuk, hogy az 1 minden számnak osztója) és $szam/2$-ig folytatjuk. Ha ugyanis egy adott szám feléig nem találtunk osztót, akkor a továbbiakban már nem is fogunk (önmagán kívül). A módosított algoritmust felhasználva akkor mondjuk, hogy egy szám prím, ha valódi osztóinak száma 0.

4.3.3. Feladatok

1. Március hónapban minden héten feljegyeztük az euró árfolyamát. Döntsük el, hogy az folyamatosan nőtt-e, azaz minden héten magasabb volt-e az előző heti értéknél.
2. Egy oktató kigyűjtötte a zh pontokat. Arra kíváncsi, volt-e hibátlan dolgozat (max 50p).
3. A rendőrség munkatársai automata sebességmérő műszert (trafipax) állítottak fel az egyik 30-as sebességkorlátozással ellátott útszakaszon. Minden nap végén szeretnék ellenőrizni, hogy volt-e aki túllépte a sebességhatárt. Írjunk ehhez algoritmust.
4. Egy adott számpárról döntsük el, hogy ikerprímek-e (prímek, melyek különbsége 2).
5. Egy adott számról döntsük el, hogy tökéletes-e. Tökéletesnek mondunk egy számot, ha egyenlő a nála kisebb osztóinak összegével. A legkisebb tökéletes szám a 6, mert $6 = 1 + 2 + 3$. A megoldáshoz felhasználhatjuk az osztók számolására írt algoritmust két módosítással: most az osztókat összegezni kell, de az összegzésbe a számot önmagát nem kell belevenni.
6. Egy adott számról döntsük el, hogy tükörszám-e (visszafelé olvasva ugyanazt az értéket kapjuk, pl.: 121). A megoldásnál tekintsük a számot karakterek sorozataként, melynek elemeit két egymásba ágyazott ciklusban vizsgáljuk. A külső ciklus indexe 1-től n-ig növekvően, míg a belső ciklus indexe n-től 1-ig csökkenően haladjon. Ha a sorozat elemei rendre megegyeznek, azaz $szam[1] = szam[n]$, $szam[2] = szam[n-1]$, ..., $szam[n] = szam[1]$, akkor a szám tükörszám. Vegyük észre, hogy elég a sorozat feléig vizsgálni az elemeket!

4.4. Kiválasztás

Adott egy N elemű sorozat és egy, a sorozat elemein értelmezett T tulajdonság.




```
1 sor[N] := [...]
2 i := 1
3 Amíg i<= N ÉS sor[i]  nem T tulajdonságú addig
4     i := i + 1
5 Ciklus vége
6 Ki: i
```

12. ábra. Egy sorozatból a T tulajdonságú elem kiválasztása

A feladat a T tulajdonsággal rendelkező elem sorszámának meghatározása. Figyelem! A megoldó algoritmus csak akkor működik, ha biztosan van T tulajdonságú elem. Emiatt a gyakorlatban a kiválasztás algoritmusát az eldöntés algoritmusával kombinálva szokták alkalmazni. Vagyis először ellenőrizni kell, hogy létezik-e a sorozatban T tulajdonságú elem. Ha igen, akkor az első ilyen elem lesz az algoritmus eredménye.

Amennyiben a sorozatban több T tulajdonságú elem is előfordulhat, szükség lehet annak specifikálására, hogy az algoritmus a sorozat hanyadik T tulajdonságú elemét szolgáltassa eredményül. Ennek a problémának általános megoldása, hogy először a számlálás programozási tételt alkalmazva meghatározzuk a T tulajdonságú elemek számát. Majd a kiválasztás algoritmusban felveszünk egy találatszámláló változót 0 kezdőértékkel, amit a ciklusmagban növelünk, és a ciklus kilépési feltételében vizsgáljuk mikor érjük el a kívánt értékét.

4.4.1. Példa

2013-ban minden héten feljegyeztük az euró árfolyamát. Hányadik héten érte el először a $300 ft$-ot? **Megoldás:** Tulajdonképpen azt kell eldönteni, hogy volt-e olyan hét amikor az euró árfolyama 300 ft felett volt és ha igen, hányadik volt ez a hét. Tehát az eldöntés algoritmusát használjuk azzal a kiegészítéssel, hogy itt a kimenet a megtalált elem sorszáma. Ha azonban nincs a feltételnek eleget tevő elem a sorozatban, ez a sorszám $N+1$, ami félrevezető lehet mert "értelmes" adatnak tűnik. Ezért az egyértelműség kedvéért a kimenet az eldöntés eredményét is tartalmazza. Így ha nem volt olyan hét 2013-ban, amikor az euró árfolyama elérte a 300 ft-ot, az algoritmus eredménye: *van=hamis, i=53*.

programkód 6. Mikor érte el az euró árfolyama a 300 ft-ot

```
Valtozok: euro_arf[N],N,van,i
N := 52
van := hamis
i := 1
euro_arf[N] := [ ... ]
Amig i<=N ES euro_arf[i]<300 addig
  i := i + 1
Ciklus vege
Ha i<=N akkor
  van := igaz
Feltetel vege
Output: van, i
```

4.4.2. Feladatok

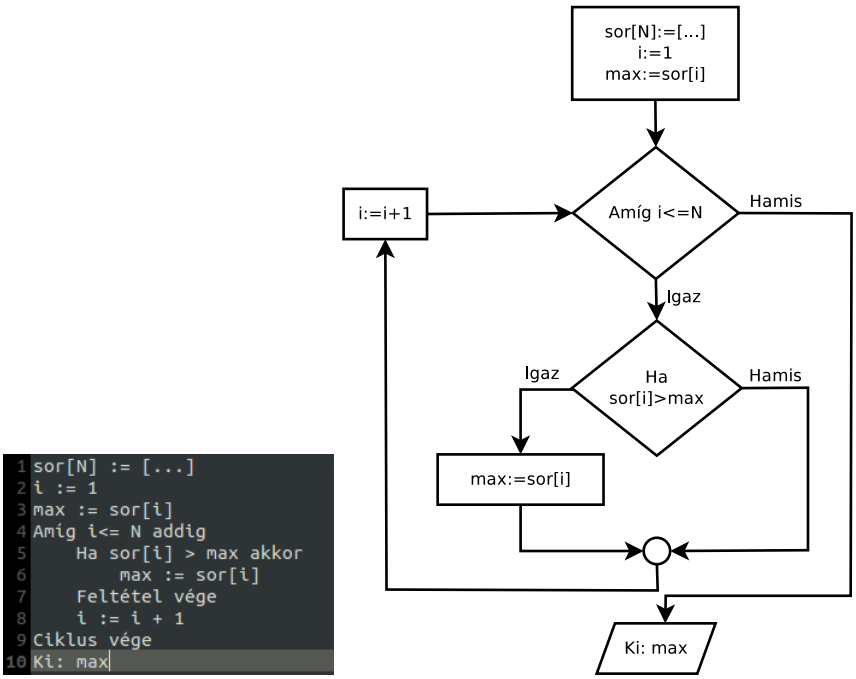
1. Januárban minden nap feljegyeztük Miskolcon az átlaghőmérsékletet. Melyik nap volt először mínusz? Melyik nap volt utoljára plusz? Figyelem! Ha nem volt ilyen nap, a kiválasztás algoritmusa önmagában véve hibás eredményre vezet.
2. Január hányadik napján van Mária-nap (ha van egyáltalán)?
3. Egy adott betűről mondjuk meg, hogy hányadik az ábécében.

4. Adottak a mérnökinformatika alapszak 2014-es felvételi ponthatárai a magyarországi egyetemekre csökkenő sorrendben. Tételezzük fel, hogy mindre beadtuk a jelentkezési lapunkat. Kérdésünk, hogy egy adott pontszámmal melyik egyetemre nyertünk felvételt. Másként megfogalmazva: melyik az az első egyetem (hányadik a rangsorban), ahol a ponthatár alacsonyabb az általunk megadott pontszámnál.

4.5. Szélsőérték kiválasztás

Egy sorozat legnagyobb vagy legkisebb elemét keressük. A kiválasztás programozási tétel alkalmazásának speciális esetei, amelyek számsorozat esetén mindig alkalmazhatók.

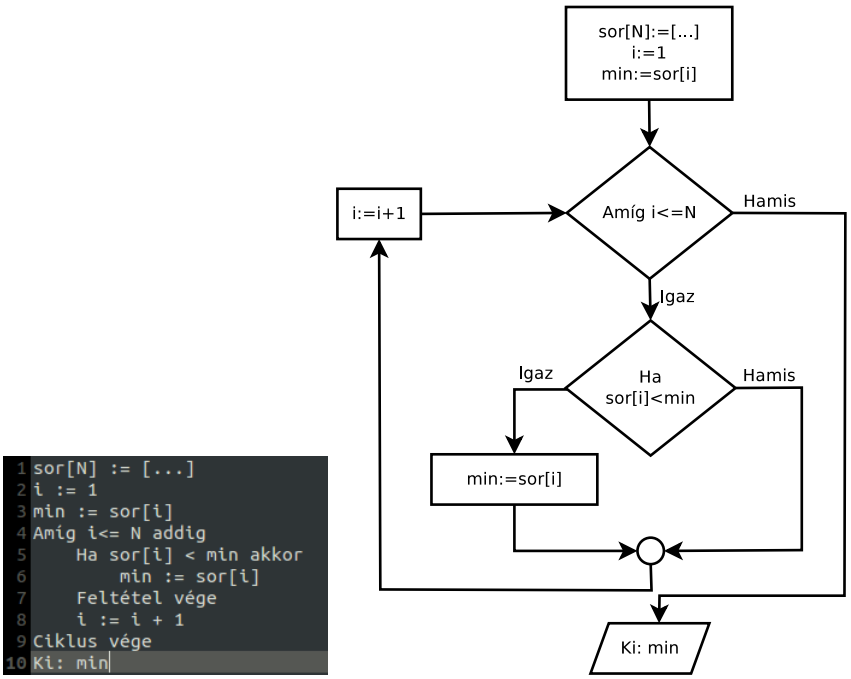
Maximum-kiválasztás: adott egy N elemű számsorozat, amelynek keressük a legnagyobb elemét. Az algoritmus leírása: vesszük a sorozat első elemét és feltesszük, hogy ez a legnagyobb (max). A sorozat többi elemét ciklusban ehhez az elemhez hasonlítjuk. Ha találunk ettől az elemtől nagyobb számot, akkor ezt megjegyezzük mint legnagyobbat (max) és a továbbiakban ehhez hasonlítjuk a többi elemet. Az algoritmus akkor ér véget, ha a számsorozatot végignéztük. Az eredmény a sorozat legnagyobb eleme (max).




```
1  sor[N] := [...]
2  i := 1
3  max := sor[i]
4  Amíg i<= N addig
5      Ha sor[i] > max akkor
6          max := sor[i]
7      Feltétel vége
8      i := i + 1
9  Ciklus vége
10 Ki: max
```

13. ábra. Egy számsorozat legnagyobb elemének kiválasztása

Minimum-kiválasztás: adott egy N elemű számsorozat, amelynek keressük a legkisebb elemét. Az algoritmus leírása: vesszük a sorozat első elemét és feltesszük, hogy ez a legkisebb (min). A sorozat többi elemét ciklusban ehhez az elemhez hasonlítjuk. Ha találunk ettől az elemtől kisebb számot, akkor ezt megjegyezzük mint legkisebbet (min) és a továbbiakban ehhez hasonlítjuk a többi elemet. Az algoritmus akkor ér véget, ha a számsorozatot végignéztük. Az eredmény a sorozat legkisebb eleme (min).



14. ábra. Egy számsorozat legkisebb elemének kiválasztása

Mindkét esetben az eredmény a sorozatban *elsőként* megtalált legnagyobb/legkisebb elem. Ha azt szeretnénk, hogy a sorozatban *utolsóként* megtalált legnagyobb/legkisebb elem legyen az eredmény, akkor vagy fordított irányban vizsgáljuk a sorozat elemeit, vagy az összehasonlításnál a nagyobb illetve kisebb reláció mellett az egyenlőséget is meg kell engedjük.

4.5.1. Példa

Augusztusban minden nap feljegyeztük a Balaton vízhőmérsékletét. Melyik nap volt a legalacsonyabb a vízhőmérséklet és mennyi volt ez az érték? Melyik nap volt a legmagasabb a vízhőmérséklet és mennyi volt ez az érték? **Megoldás:** A maximum- és minimum-kiválasztás algoritmusait egyszerre alkalmazzuk azzal a kiegészítéssel, hogy nemcsak a sorozat legnagyobb ill. legkisebb elemére, hanem azok helyére is kíváncsiak vagyunk (arra, hogy hányadikak a sorozatban). Az algoritmus eredménye a sorozatban elsőként megtalált legnagyobb és legkisebb elem és azok helye.

programkód 7. Legkisebb és legnagyobb hőmérséklet-érték meghatározása

```
Valtozok:min-nap,min-ertek,max-nap,max-ertek,i,N,vizhom[N]
N := 31
vizhom[31] := [ ... ]
min-nap := 1
min-ertek := vizhom[min-nap]
max-nap := 1
max-ertek := vizhom[max-nap]
i := 1
```

```
Amig i<=N addig
  Ha vizhom[i] < min-ertek akkor
      min-nap := i
      min-ertek := vizhom[i]
  Feltetel vege
  Ha vizhom[i] > max-ertek akkor
      max-nap := i
      max-ertek := vizhom[i]
  Feltetel vege
  i := i+1
Ciklus vege
Output: min-nap, min-ertek, max-nap, max-ertek
```

4.5.2. Feladatok

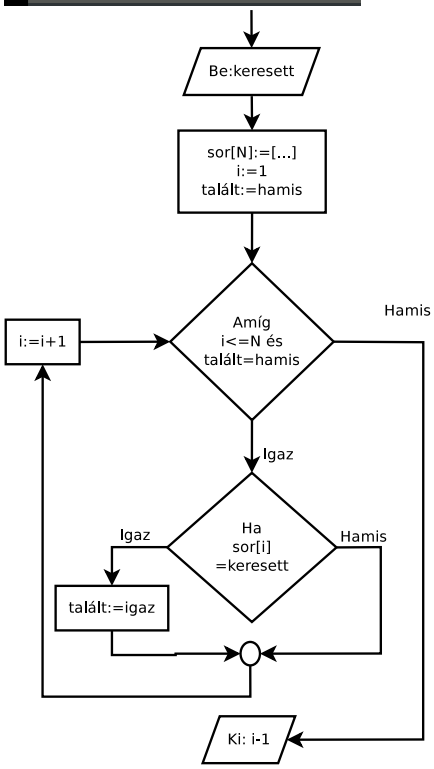
1. Adott a tanulók félév végi átlagának sorozata. Mi volt a legmagasabb és a legalacsonyabb tanulmányi átlag?
2. Egy oktató zh statisztikát készít. Szeretné megtudni, hogy mi volt az elért legmagasabb pontszám és hány ilyen dolgozat született. Figyelem! Itt először a maximum-kiválasztás, majd a számlálás algoritmusát kell alkalmazni.
3. A múlt hónapban minden nap feljegyeztem az euró árfolyamát. Melyik nap volt a legmagasabb és melyik nap volt a legalacsonyabb az árfolyam, és mennyi volt az értéke?
4. Keressük meg egy megadott szám legnagyobb osztóját, ami nem önmaga.
5. Egy N elemű számsorozatból válasszuk ki a legkisebb páros számot. Figyelem! Itt a minimum-kiválasztás algoritmusának alkalmazása előtt ki kell választani a sorozatból az első páros számot. A minimum-kiválasztás során ez az érték lesz a kiinduló. A feladat másik lehetséges megoldását a kiválogatás tételnél tárgyaljuk.
6. Adott az Alpok csúcsainak magasságát tároló sorozat. Határozza meg, hogy a 3000 méteres csúcsok közül melyik a legnagyobb. A feladat általános megoldása, hogy először kiválogatjuk a 3000 méteres csúcsokat (lásd 5.1 kiválogatás tétel), majd erre a sorozatra alkalmazzuk a maximum-kiválasztás tételét. Ennek ismerete nélkül próbáljunk más megoldást találni.

4.6. Lineáris keresés

Adott egy N elemű sorozat és egy, a sorozaton értelmezett T tulajdonság. A feladat megkeresni a sorozatban egy adott T tulajdonságú elemet. A legegyszerűbb kereső algoritmus, amely a teljes sorozatot végignézi és minden elemére megvizsgálja a feltétel teljesülését. Tulajdonképpen az eldöntés és a kiválasztás tételének együttes alkalmazása, azaz ez az algoritmus akkor is működik, ha nincs a sorozatban T tulajdonságú elem.

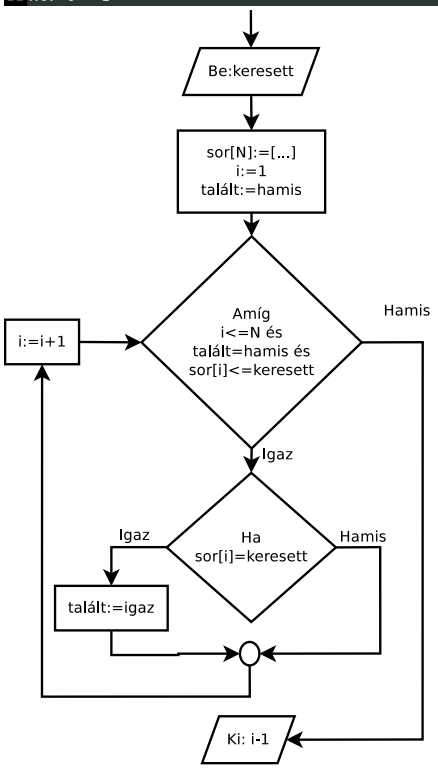
Mivel teljes keresésről van szó, érdemes szétválasztani a rendezett sorozat és a rendezetlen sorozat esetét. Sikeres keresés esetén a keresési lépések száma mindkét eljárás során egyenesen arányos a sorozat elemszámával (annak lineáris függvénye, átlagosan $elemszam/2$); ám sikertelen keresés esetén a rendezett sorozatban kereső algoritmus gyorsabb, mint a rendezetlen sorozatban kereső algoritmus.

```
Be: keresett
sor[N] := [...]
i := 1
talált := hamis
Amig i<= N ÉS talált == hamis addig
    Ha sor[i] == keresett akkor
        talált := igaz
    Feltétel vége
    i := i + 1
Ciklus vége
Ki: i - 1
```



15. ábra. Lin. keresés rendezetlen sorozatban

```
Be: keresett
sor[N] := [...]
i := 1
talált := hamis
Amig i<= N ÉS talált == hamis ÉS sor[i] <= keresett addig
    Ha sor[i] == keresett akkor
        talált := igaz
    Feltétel vége
    i := i + 1
Ciklus vége
Ki: i - 1
```



16. ábra. Lineáris keresés rendezett sorozatban

A 16. ábrán látható algoritmus növekvően rendezett sorozat esetén működik. Ha a sorozat csökkenő sorrendű, addig van értelme a keresésnek amíg a keresett elem kisebb vagy egyenlő a sorozat vizsgált eleménél.

4.6.1. Példa

A lineáris keresés programozási tétel alkalmazásával döntsük el egy adott n számról, hogy prímszám-e. **Megoldás:** A megadott számnak keressük az osztóit. Ha találunk 1-től és önmagától különböző osztót, akkor nem prím. A tanult algoritmust alkalmazva, először feltesszük, hogy az adott számnak nincs osztója. Ezután ciklusban tekintjük a számokat 2-től $(n-1)$-ig és keressük a megadott szám egy osztóját. Ha találunk osztót, akkor nem prímszám. Ha nem találunk, akkor prím. A keresés gyorsítható, ha végiggondoljuk, hogy felesleges $(n-1)$-ig keresni, elég $(n/2)$-ig. Ha ugyanis egy n számnak 2 nem osztója, akkor $(n/2)$ sem az; ha 3 nem osztója, akkor $(n/3)$ sem az; és így tovább.

programkód 8. Prímszám-e?

```
Valtozok: prim, van_oszto, oszto, szam
Input: szam
prim := igaz
van_oszto := hamis
oszto := 2;
Amig oszto<szam/2 ES van_oszto=hamis addig
   Ha szam mod oszto = 0 akkor
     van_oszto := igaz
   Feltetel vege
   oszto := oszto + 1;
Ciklus vege
Ha van_oszto akkor
   prim := hamis
Output: prim
```

4.6.2. Példa

Számoljuk meg hány prímszám van N-ig. **Megoldás:** A lineáris keresés és a megszámlálás tételének kombinálása.

programkód 9. Prímszámok száma N-ig

```
Valtozok: db, i, N, van_oszto, oszto
Input: N
db := 0
i := 1
Amig i<=N addig
  /* osztok szekvencialis keresese */
  van_oszto := hamis
  oszto := 2
  Amig oszto<szam ES van_oszto=hamis addig
    Ha szam mod oszto = 0 akkor
      van_oszto := igaz
    Feltetel vege
    oszto := oszto + 1
  Ciklus vege
  /************************/
  /* osztok szamlalasa */
  Ha van_oszto=hamis akkor
    db := db+1
  Feltetel vege
  i := i + 1
Ciklus vege
Output: db
```

4.6.3. Feladatok

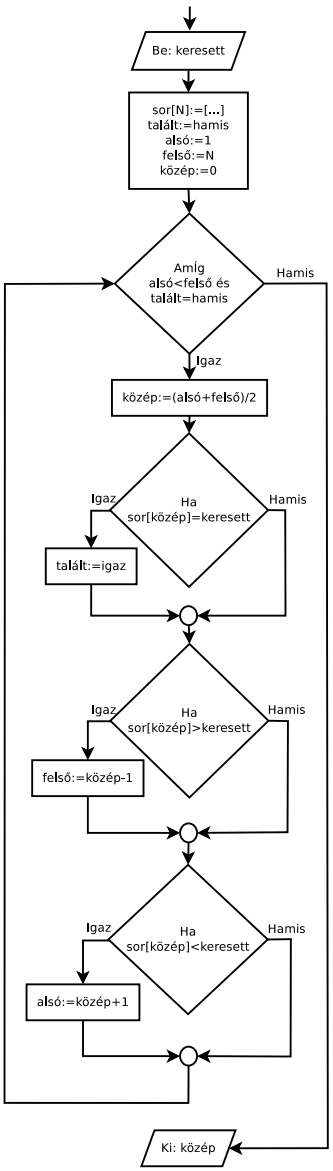
1. Adjuk meg a prímszámokat kereső Eratoszthenészi-szita algoritmust. A módszer a következő. Felírjuk a számokat N-ig. 2 prímszám, mert nincs nála kisebb osztója 1-en kívül, ezt megjegyezzük. Kihúzzuk a többszöröseit, mivel azok nem prímszámok. Ezután 3 a következő, ami még nincs kihúzva. A nála kisebb összes szám többszöröseit kihúztuk, tehát prímszám. A többszörösei viszont nem prímek: kihúzzuk az összes 3-mal oszthatót. 4-et már kihúztuk. 5 a következő prím, kihúzzuk a többszöröseit, és így tovább.
2. Adjuk meg azt az algoritmust, ami megkeresi az összes 3 jegyű tükörszámot (olyan számok, amelyek megegyeznek tükörképükkel).
3. Keressük meg N-ig az ikerprímeket. Ikerprímnek nevezzük két olyan prímszám együttesét, amelyek 2-vel térnek el egymástól: például 5 és 7.
4. Keressük meg N-ig a tökéletes számokat. Egy adott szám tökéletes, ha egyenlő a nála kisebb osztóinak összegével (keresés és összegzés). Például $6 = 1 + 2 + 3$.
5. Keressük meg N-ig a barátságos számpárokat. Két szám barátságos, ha az egyik egyenlő a másik önmagánál kisebb osztóinak összegével és viszont, például $(220, 284)$.
6. Keressünk N-jegyű Armstrong számokat. Armstrong számnak nevezzük azt a számot, melynek számjegyei N-dik hatványainak összege éppen a számot adja. Például: négyjegyű Armstrong-szám a 1634, mivel $1634 = 1^4 + 6^4 + 3^4 + 4^4$.

4.7. Logaritmikus keresés

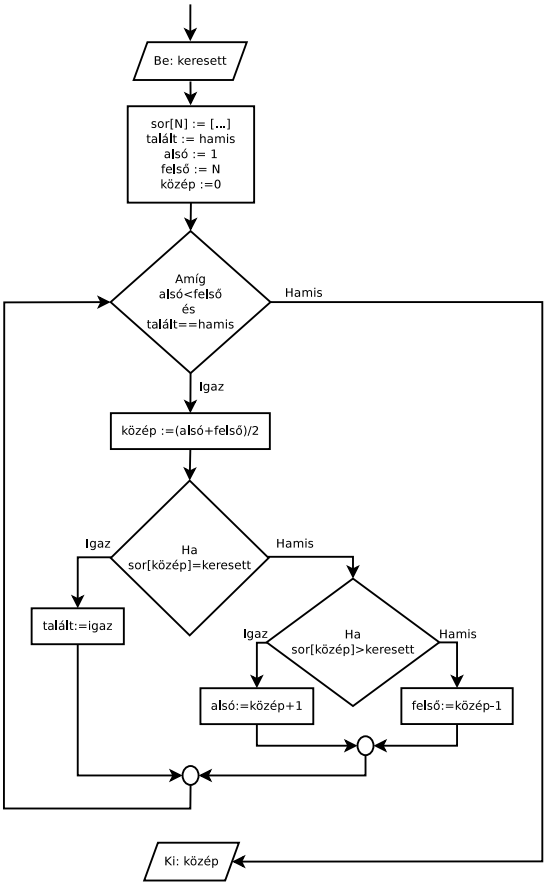
Adott egy N elemű rendezett sorozat és egy, a sorozaton értelmezett T tulajdonság. A feladat megkeresni a sorozatban egy adott T tulajdonságú elemet. A lineáris kereső eljáráshoz hasonlóan az algoritmus akkor is működik, ha nincs a sorozatban T tulajdonságú elem. Azért hívják logaritmikus keresésnek, mert az összehasonlítások száma az elemszám kettes alapú logaritmusával arányos $log_2 N$, ami nagy elemszámok esetén lényegesen kisebb lehet, mint a lineáris keresés futási ideje. Vagyis sokkal hatékonyabb mint a lineáris keresés, de csak rendezett sorozatra alkalmazható. A megoldás során alkalmazott módszer intervallumfelezésen alapul, ezért bináris vagy felező keresésnek is nevezzük.

```
Be: keresett
sor[N] := [...]
talált := hamis
alsó   := 1
felső  := N
közép  := 0
Amíg alsó < felső ÉS talált == hamis addig
    közép := (alsó+felső)/2 egészrésze
    Ha sor[közép] == keresett akkor
        talált := igaz
    Feltétel vége
    Ha sor[közép] > keresett akkor
        felső := közép - 1
    Feltétel vége
    Ha sor[közép] < keresett akkor
        alsó := közép + 1
    Feltétel vége
Ciklus vége
Ki: közép
```

```
Be: keresett
sor[N] := [...]
talált := hamis
alsó   := 1
felső  := N
közép  := 0
Amíg alsó < felső ÉS talált == hamis addig
    közép := (alsó+felső)/2 egészrésze
    Ha sor[közép] == keresett akkor
        talált := igaz
    Feltétel vége
    Ha sor[közép] > keresett akkor
        alsó := közép + 1
    Egyéként
        felső := közép - 1
    Feltétel vége
Ciklus vége
Ki: közép
```



17. ábra. Logaritmikus keresés rendezett növekvő sorozatban



18. ábra. Logaritmikus keresés rendezett csökkenő sorozatban

Első lépésben a keresési teret megfelezzük. Ha nem a középső a keresett elem, akkor megvizsgáljuk, hogy az intervallum alsó vagy felső felébe esik-e. Növekvő sorozat esetén, ha az alsó felébe esik, akkor a kiinduló intervallum felső határát lejjebb csúsztatjuk, a középső elem alá. Ha a felső felébe esik, akkor viszont a kiinduló intervallum alsó határát kell feljebb mozgatni, a középső elem fölé. Csökkenő sorozat esetén, ha az alsó felébe esik, akkor a kiinduló intervallum alsó határát feljebb csúsztatjuk, a középső elem fölé. Ha a felső felébe esik, akkor viszont a kiinduló intervallum felső határát kell lejjebb mozgatni, a középső elem alá. A következő iteráció során az ilymódon megfelezett intervallum közepét tekintjük és megnézzük, hogy a keresett elem az intervallum alsó vagy felső felébe esik-e. Ezt addig folytatjuk, amíg meg nem találjuk a keresett elemet, vagy a keresési intervallum tovább már nem felezhető.

4.7.1. Példa

Adott a Forma-1 versenyzők 2013-as tabellája (feltesszük hogy létezik és nem üres). Hányadik a ponttáblázatban az a versenyző, aki 100 pontot gyűjtött (ha van ilyen versenyző)? Mivel a bemenő számsorozat csökkenő sorrendben rendezett, a lineáris és a logaritmikus keresési tétel alkalmazásával is megoldható a feladat.

programkód 10. Lineáris keresés a Forma-1 tabellán

```
Valtozok: i, keresett, talalt, pont[N], N, hely
keresett := 100
N := 23
pont[N] := [397,242,199,189,183,171,132,112,73, ..., 0]
talalt := hamis, hely := 0
i := 0
Ciklus
  i := i + 1
  Ha pont[i] = keresett akkor
    talalt := igaz
    hely := i
  Feltetel vege
Amig i<N ES talalt = hamis ES pont[i] > keresett
Output: hely
```

Ha ismerjük a lista adatainak eloszlását, a keresés irányának megfelelő megválasztásával (1-től N-ig, vagy fordítva N-től 1-ig) befolyásolhatjuk a lineáris keresés gyorsaságát.

programkód 11. Logaritmikus keresés a Forma-1 tabellán

```
Valtozok: talalt, also, felso, kozepso, keresett, hely
keresett := 100
N := 23
pont[N] := [397,242,199,189,183,171,132,112,73, ..., 0]
talalt := hamis, hely := 0
also := 1, felso := N
Amig also < felso ES talalt = hamis addig
  kozepso := (also + felso)/2 egeszresze
  Ha pont[kozepso] = keresett akkor
    talalt := igaz
    hely := kozepso
  Feltetel vege
  Ha pont[kozepso] < keresett akkor
    felso = kozepso - 1
  Feltetel vege
  Ha pont[kozepso] > keresett akkor
    also = kozepso + 1
  Feltetel vege
Ciklus vege
Output: hely
```

A fenti példa megoldásakor a lineáris kereső algoritmus 9 ciklus végrehajtása után áll le azzal az eredménnyel, hogy nincs ilyen pontszámú versenyző a tabellán (hely=0), míg a logaritmikus kereső 4 ciklus után jut ugyanerre az eredményre.

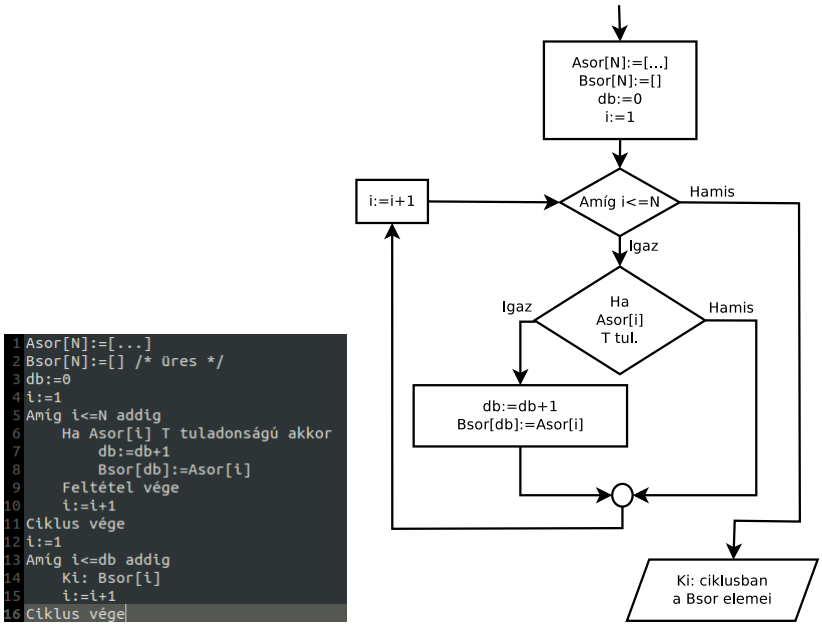
4.7.2. Feladatok

1. Egy adott betűről mondjuk meg, hányadik az angol ABC-ben (26 karakter).
2. Adottak a Miskolci Egyetem GÉIK mérnök informatikus alapszakra 2014-ben felvételizők pontszámai (300 fő) csökkenő sorrendben. Hányan nyertek felvételt, ha 270 pont a felvételi ponthatár?
3. Egy oktató a zh dolgozatokat (150 darab) az elért pontszám alapján növekvő sorrendbe rakta. Arra kíváncsi, hogy hányan nem teljesítették a minimum szintet (az 50 pontos dolgozat 50%-a).
4. A Statisztikai Hivatal havi összesítésben tartja nyilván a születések számát. 2013-ban hányadik hónapban érte el ez a szám a 40 ezret? Figyelem! Az eredeti számsorozat nem biztos hogy monoton növekvő, de az aggregált sorozat biztosan az. Erre kell alkalmazni a keresési algoritmust.

5. Egy sorozathoz egy sorozatot rendelő algoritmusok

5.1. Kiválogatás

Adott egy N elemű A sorozat és egy, a sorozat elemein értelmezett T tulajdonság. A feladat az A sorozat összes T tulajdonsággal rendelkező elemének kigyűjtése egy B sorozatba. Az algoritmus lényege, hogy az A sorozat elemeit egyenként tekintve mindegyikről el kell dönteni, hogy rendelkezik-e a T tulajdonsággal. Ha igen, kap egy sorszámot és bekerül az új B sorozatba.




```
1  Asor[N]:=[...]
2  Bsor[N]:=[] /* Üres */
3  db:=0
4  i:=1
5  Amíg i<=N addig
6      Ha Asor[i] T tuladonságú akkor
7          db:=db+1
8          Bsor[db]:=Asor[i]
9      Feltétel vége
10     i:=i+1
11 Ciklus vége
12 i:=1
13 Amíg i<=db addig
14     Ki: Bsor[i]
15     i:=i+1
16 Ciklus vége
```

19. ábra. Egy sorozat T tulajdonságú elemeinek kiválogatása

5.1.1. Példa

Gyűjtsük ki N-ig az összes négyzetszámot. **Megoldás:** A top-down (felülről lefelé történő) algoritmus tervezési elvet követve első közelítésben azt mondjuk, hogy ha az aktuálisan vizsgált szám négyzetszám, akkor kap egy sorszámot és bekerül a négyzetszámokat tároló sorozatba. Ezt a tömböt az algoritmus végén elemenként íratjuk ki. Annak eldöntése, hogy egy szám négyzetszám-e, külön eljárás feladata.

programkód 12. Négyzetszámok kigyűjtése N-ig

```
  Valtozok: N, negyzetszam[N], i, db
  Input: N
  i := 1
  db := 0
  Amig i<=N addig
    Ha negyzetszam(i) akkor
      db := db + 1
      negyzetszam[db] := i
    Feltetel vege
    i := i+1
  Ciklus vege
  i := 1
  Amig i<=db addig
    Output: negyzetszam[i]
    i := i+1
  Ciklus vege

Eljaras: negyzetszam
  Valtozok: i, szam, kilep
  Input: szam
  i := 1
  kilep := hamis
  Amig szam>=i*i ES kilep=hamis addig
    Ha szam=i*i akkor
      kilep=igaz
    Egyebkent
      i := i+1
    Feltetel vege
  Ciklus vege
  Output: kilep
Eljaras vege
```

5.1.2. Példa

N elemű számsorozatból válasszuk ki a legkisebb páros számot. A feladat egyik megoldása, hogy először kiválogatjuk a páros számokat az adott számsorozatból, majd erre a sorozatra alkalmazzuk a minimum-kiválasztás algoritmusát.

programkód 13. Egy sorozatban a legkisebb páros szám megkeresése (kiválogatással)

```
Valtozok: i,db,N,szamsor[N],paros[N],min,talalt
szamsor[N] := [ ... ]
i := 1
db := 0
Amig i<=N addig
  Ha szamsor[i] mod 2 = 0 akkor
    db := db + 1
    paros[db] := i
  Feltetel vege
  i := i+1
Ciklus vege
i := 1
Ha db>0 akkor
  min := paros[i]
  van_paros := igaz
Egyebkent
  min := 0
  van_paros := hamis
Feltetel vege
Amig i<=db addig
  Ha paros[i] < min akkor
     min := paros[i]
  Feltetel vege
  i := i+1
Ciklus vege
Output: van_paros, min
```

A feladat másik megoldása, hogy először kiválasztjuk a sorozatban az első páros számot. Ez lesz a minimum-kiválasztás algoritmusában a kiinduló érték. Erre azért van szükség, mert ha a teljes sorozatra alkalmazzuk a minimum-kiválasztás algoritmusát előfordulhat, hogy a legkisebb érték nem páros.

programkód 14. Egy sorozatban a legkisebb páros szám megkeresése

```
Valtozok: i,N,szamsor[N],min,elso_paros,van_paros
szamsor[N] := [ ... ]
i := 1
elso_paros := 0
van_paros := hamis
Amig i<=N ES szamsor[i] mod 2 != 0 addig
  i := i+1
Ciklus vege
van_paros := (i<=N)
Ha van_paros akkor
  elso_paros := szamsor[i]
Feltetel vege
```

```
min := elso_paros
i := 1
Amig i <= N addig
  Ha min > szamsor[i] ES szamsor[i] mod 2 = 0 akkor
    min := szamsor[i]
  Feltetel vege
  i := i+1
Ciklus vege
Output: van_paros, min
```

5.1.3. Feladatok

1. Az angol ABC-ből válogassuk ki a magánhangzókat.
2. A zh dolgozatok közül válogassuk ki a 0 pontosakat.
3. Gyűjtsük ki N-ig a prímszámokat.
4. Gyűjtsük ki az első N darab négyzetszámot.

5.2. Rendezések

Az algoritmusok témakörében a rendező eljárások külön fejezetet alkotnak. A rendezés alapfeladata szerint adott egy N elemű rendezetlen A sorozat, és feladatunk a sorozat elemeinek növekvő (vagy csökkenő) sorrendbe állítása. Ez nagy műveletigényű feladat. Az egyes módszerek hatékonyságának vizsgálata a megoldás során alkalmazott elem-összehasonlítások és mozgatások számának összevetését jelenti. Ebben a szakaszban a legalapvetőbb algoritmusokat tárgyaljuk, amelyek N^2-el arányos műveletigényűek. Ezek megértése feltétlenül szükséges a hatékonyabb (rendszerint csak speciális előfeltételek teljesülése esetén alkalmazható) algoritmusok későbbi vizsgálatához [10], [11].

5.2.1. Rendezés közvetlen kiválasztással

programkód 15. Cserés rendezés

```
Asor[N] := [ ... ]
i := 1, j := 1, csere := 0
Amig i<=N-1 addig
  j := i+1
  Amig j<=N addig
    /* Csokkeno sorrend: Ha Asor[j] > Asor[i] akkor ... */
    /* Novekvo sorrend */
    Ha Asor[j] < Asor[i] akkor
      csere := Asor[j]
      Asor[j] := Asor[i]
      Asor[i] := csere
    Feltetel vege
    j := j+1
  Ciklus vege
  i := i+1
Ciklus vege
```

Ennél a módszernél első lépésként vesszük a sorozat első elemét. Ezt összehasonlítjuk a többi elemmel. Ha az elemeket növekvő sorrendbe állítjuk, akkor azt kell vizsgálni, hogy kisebbet találunk-e az első elemnél, mert ebben az esetben helyet cserélnek. Ezután az első helyre került elemmel hasonlítom össze a hátralévő elemeket. Így az első kör végére a legkisebb elem lesz az első helyen. A következő körben a sorozat második elemétől indulva hasonlítjuk össze az elemeket, és keressük a második helyre kerülő (második) legkisebb elemet, és így tovább. $N-1$ kör után lesz rendezett a sorozat. Ha a sorozat elemeit csökkenő sorrendbe kívánjuk állítani, akkor az elemek összehasonlításánál az a cél, hogy az aktuálisan vizsgált helyre a lehető legnagyobb elem kerüljön. Az algoritmus során az összehasonlítások száma $N*(N-1)/2$, a mozgatások száma pedig legfeljebb $3*N*(N-1)/2$.

5.2.2. Rendezés szélsőérték kiválasztással

Az előző módszer javított változata. A cél a felesleges cserék kiküszöbölése. Ezért minden körben növekvő sorozat esetén a legkisebb elemet, csökkenő sorozat esetén pedig a legnagyobb elemet választjuk ki. Ehhez két segédváltozó bevezetése szükséges. Az algoritmus során az összehasonlítások száma $N*(N-1)/2$ (megegyezik az előző módszer esetén számítottal), míg a mozgatások száma maximum $3*(N-1)+(N*N/4)$, ami egy $N/2$-es szorzóval kevesebb mint a közvetlen kiválasztásos rendezésnél.

programkód 16. Minimum-kiválasztásos rendezés

```
Asor[N] := [ ... ]
i := 1
j := 1
min := 0
minindex := 0
Amig i<=N-1 addig
  minindex := i
  min := Asor[i]
  j := i+1
  Amig j<=N addig
    /* Csokkeno sorrend eseten maximum-kivalasztas */
    /* Ha Asor[j] > max akkor ... */
    /* Novekvo sorrend */
    Ha Asor[j] < min akkor
      min := Asor[j]
      minindex := j
    Feltetel vege
    j := j+1
  Ciklus vege
  Asor[minindex] := Asor[i]
  Asor[i] := min
  i := i+1
Ciklus vege
```

5.2.3. Buborékos rendezés

A közvetlen kiválasztásos rendezéshez hasonló, csak ennél a módszernél a sorozat végéről indulva hasonlítjuk össze az elemeket. Ezért az algoritmus során az összehasonlítások és a mozgatások száma is ugyanannyi, mint a közvetlen kiválasztásos rendezésnél.

programkód 17. Buborék rendezés

```
Asor[N] := [ ... ]
i := N
j := 1
csere := 0
Amig i>=2 addig
  j := 1
  Amig j<=i-1 addig
    /* Csokkeno sorrend: */
    /* Ha Asor[j] < Asor[j+1] akkor ... */
    /* Novekvo sorrend */
    Ha Asor[j] > Asor[j+1] akkor
      csere := Asor[j]
      Asor[j] := Asor[j+1]
      Asor[j+1] := csere
    Feltetel vege
    j := j+1
  Ciklus vege
  i := i-1
Ciklus vege
```

5.2.4. Beszúró rendezés

programkód 18. Beszúró rendezés

```
Asor[N] := [ ... ]
i := 2
j := 1
seged := 0
Amig i<=N addig
  j := i-1
  seged := Asor[i]
  Amig j>0 ES seged<Asor[j] addig
      Asor[j+1] := Asor[j]
      j := j-1
  Ciklus vege
  Asor[j+1] := seged
  i := i+1
Ciklus vege
```

Ez a módszer a kártyalapok felvételét modellezi. Első körben egy lap van a kezünkben (a sorozat első eleme). A következő elemet ehhez hasonlítva helyezzük el (elé vagy utána). A harmadik elemnek is megkeressük a helyét, ide illesztjük be és így tovább. Az összehasonlítások száma legfeljebb $N*(N+1)/2-1$, a mozgatások száma pedig maximum $2*(N-1)+N*(N-1)/2$. Ennek a módszernek a javított változata a Shell-féle rendezés [12].

5.2.5. Számláló rendezés

Az egyszerű cserés (közvetlen kiválasztásos) rendezés mozgatásainak száma N-re csökkenthető ezzel az eljárással. A módszer lényege, hogy ne cserélgessük az elemeket, hanem csak számoljuk meg mindegyikre, hogy hány nála kisebb, illetve az őt megelőzők között hány vele egyenlő van (önmagát is beleértve) – azaz kapjon egy sorszámot. Ezután a sorozatot a kialakított sorrendben elemenként átmásoljuk egy új sorozatba.

programkód 19. Cserés rendezés

```
Asor[N] := [ ... ]
Bsor[N] := [ ]
i := 1
Amig i<=N addig
  Index[i] := 1
  i := i+1
Ciklus vege
i := 1
j := 1
Amig i<=N-1 addig
  j := i+1
  Amig j<=N addig
    /* Csokkeno sorrend: */
    /* Ha Asor[j] > Asor[i] akkor ... */
    /* Novekvo sorrend */
    Ha Asor[j] < Asor[i] akkor
      Index[i] := Index[i] + 1
    Egyebkent
      Index[j] := Index[j] + 1
    Feltetel vege
    j := j+1
  Ciklus vege
  i := i+1
Ciklus vege
i := 1
Amig i<=N addig
  Bsor[Index[i]] := Asor[i]
  i := i+1
Ciklus vege
```

5.2.6. Példa

Tekintsük az alábbi $N = 5$ elemű rendezetlen A sorozatot: $5, 7, 3, 1, 9$. Végezzük el a sorozat növekvő rendezését a tanult öt algoritmust felhasználva. Az egyes iterációs lépésekben történt mozgatásokat, ill. cseréket és a kialakult új sorozatokat az 1. és 2. táblázat foglalja össze.

1. táblázat. Sorozat rendezésének lépései 1.

Eljárás	i	j	mozgatás/csere	új sorozat
Közvetlen kiválasztás	1	2		
	1	3	A[3]=5, A[1]=3	3,7,5,1,9
	1	4	A[4]=3, A[1]=1	1,7,5,3,9
	1	5		
	2	3	A[3]=7, A[2]=5	1,5,7,3,9
	2	4	A[4]=5, A[2]=3	1,3,7,5,9
	2	5		
	3	4	A[4]=7, A[3]=5	1,3,5,7,9
	3	5		
	4	5		
Minimum-kiválasztás	1		min=1,minindex=4, A[1]=1, A[4]=5	1,7,3,5,9
	2		min=3,minindex=3, A[2]=3, A[3]=7	1,3,7,5,9
	3		min=5,minindex=4, A[3]=5, A[4]=7	1,3,5,7,9
	4			
Buborék rendezés	5	1		
	5	2	A[2]=3, A[3]=7	5,3,7,1,9
	5	3	A[3]=1, A[4]=7	5,3,1,7,9
	5	4		
	4	1	A[1]=3, A[2]=5	3,5,1,7,9
	4	2	A[2]=1, A[3]=5	3,1,5,7,9
	4	3		
	3	1	A[1]=1, A[2]=3	1,3,5,7,9
	3	2		
	2	1		
Beszúró rendezés	2	1	A[2]=7	5,7
	3	2	A[3]=7	
	3	1	A[2]=5	3,5,7
	4	3	A[4]=7	
	4	2	A[3]=5	
	4	1	A[2]=3	1,3,5,7
	5	4	A[5]=9	
	5	3	A[4]=7	
	5	2	A[3]=5	
	5	1	A[2]=3	1,3,5,7,9

Ebből jól látszik, hogy a közvetlen kiválasztás és a buborék rendezés azonos műveletigényű eljárások, az adott feladatot 5 cserével oldották meg. Ehhez képest a minimum-kiválasztáson alapuló rendezés csak 3 cserét hajtott végre a sorozatban. A beszúró rendezés $N - 1$ (itt 4)

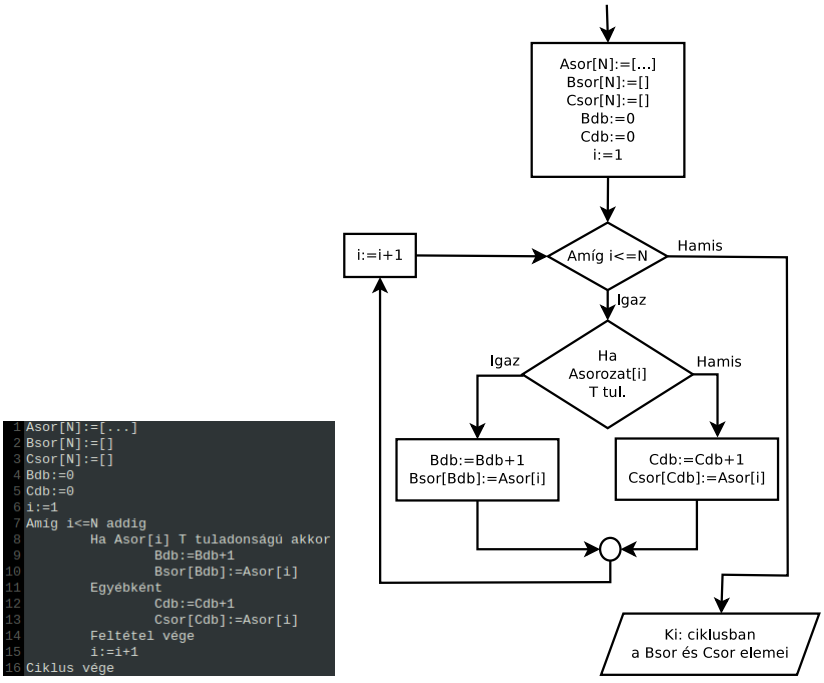
lépésben oldja meg a feladatot, míg a számláló rendezés a kijelölő indexek felhasználásával a teljes sorozatot rendezetten másolja át egy új sorozatba, azaz mindig N mozgatás történik.

2. táblázat. Sorozat rendezésének lépései 2.

Eljárás	i	j	mozgatás/csere	új sorozat
Számláló rendezés	1	2	Index[2]=2	
	1	3	Index[1]=2	
	1	4	Index[1]=3	
	1	5	Index[5]=2	
	2	3	Index[2]=3	
	2	4	Index[2]=4	
	2	5	Index[5]=3	
	3	4	Index[3]=2	
	3	5	Index[5]=4	
	4	5	Index[5]=5	
	1		B[Index[1]=3]=A[1]=5	
	2		B[Index[2]=4]=A[2]=7	
	3		B[Index[3]=2]=A[3]=3	
	4		B[Index[4]=1]=A[4]=1	
	5		B[Index[5]=5]=A[5]=9	1,3,5,7,9

6. Egy sorozathoz több sorozatot rendelő algoritmus

6.1. Szétválogatás




```
Asor[N]:=[...]
Bsor[N]:=[]
Csor[N]:=[]
Bdb:=0
Cdb:=0
i:=1
Amíg i<=N addig
        Ha Asor[i] T tuladonságú akkor
                Bdb:=Bdb+1
                Bsor[Bdb]:=Asor[i]
        Egyébként
                Cdb:=Cdb+1
                Csor[Cdb]:=Asor[i]
        Feltétel vége
        i:=i+1
Ciklus vége
```

20. ábra. Egy sorozat elemeinek szétválogatása két csoportba

Adott egy N elemű A sorozat és egy, a sorozat elemein értelmezett T tulajdonság. Az alapfeladat a sorozat szétválogatása két csoportba: T tulajdonságú elemek és a T tulajdonsággal nem rendelkező elemek. Hasonló a kiválogatás tételéhez, csak itt két új sorozatra (B és C) van szükség. A megoldás során sorra vesszük az adott A sorozat elemeit és mindegyikről eldöntjük, hogy rendelkezik-e a T tulajdonsággal. Ha igen, kap egy sorszámot és betesszük az elemet a B sorozatba. Ha nem, kap egy sorszámot és a C sorozatba kerül.
A tétel alkalmazásának további változataiban a T tulajdonság olyan, ami alapján az eredeti sorozat elemeit több csoportba tudjuk szétválogatni.

6.1.1. Példa

Egy adott N elemű számhalmazt válogassunk szét két csoportra: gyűjtsük külön a pozitív, illetve a negatív számokat.

programkód 20. N elemű számhalmaz szétválogatása pozitív és negatív számokra

```
Valtozok: i,N,szamsor[N],Psorozat[N],Nsorozat[N],Pdb,Ndb
szamsor[N] := [ ... ]
Pdb := 0
Ndb := 0
i := 1
Amig i<=N addig
  Ha szamsor[i] >=0 akkor
    Pdb := Pdb+1
    Psorozat[Pdb] := szamsor[i]
  Egyebkent
    Ndb := Ndb+1
    Nsorozat[Ndb] := szamsor[i]
  Feltetel vege
  i := i+1
Ciklus vege
i := 1
Amig i<=Pdb addig
  Ki: Psorozat[i]
  i := i+1
Ciklus vege
i := 1
Amig i<=Ndb addig
  Ki: Nsorozat[i]
  i := i+1
Ciklus vege
```

6.1.2. Feladatok

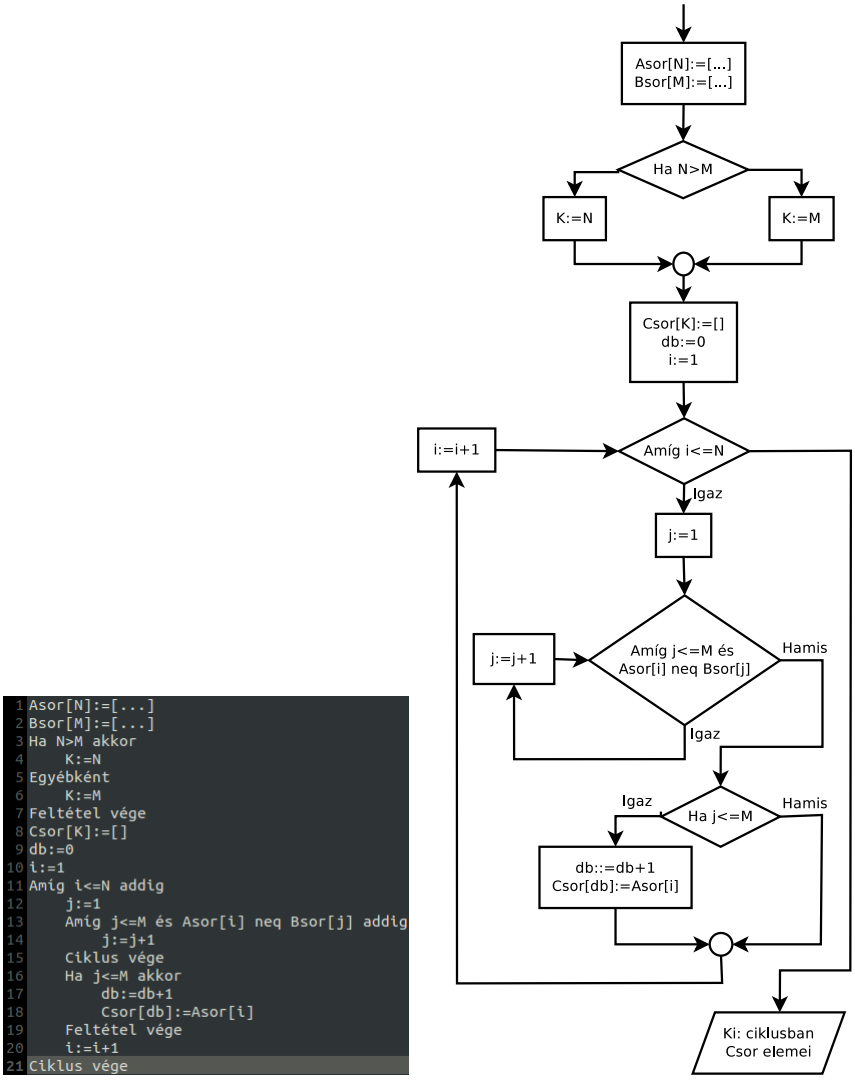
1. Egy N elemű számhalmaz elemeit válogassuk szét két csoportba: prím számok és nem prímek.

2. Egy oktató 5 csoportba válogatja szét a vizsgadolgozatokat az elért pontszámok alapján: 0-19 pontig elégtelen, 20-26 pontig elégséges, 27-32 pontig közepes, 33-36 pontig jó, 37-40 pontig jeles. Szeretné megtudni, hogy hány dolgozat került az egyes csoportokba.
3. Egy táncversenyen a díjakat az alábbi pontozás alapján határozzák meg: 33-35 pont 3. díj, 36-38 pont 2. díj, 39-40 pont 1. díj. A 25 induló csapatból gyűjtse össze az 1. , 2. és 3. díjas csapatok sorszámát.

7. Több sorozathoz egy sorozatot rendelő algoritmusok

7.1. Metszet

Adott egy N elemű A sorozat, és egy M elemű B sorozat. A feladat a két sorozat azonos elemeinek kiválogatása egy harmadik C sorozatba.




```
1  Asor[N]:=[...]
2  Bsor[M]:=[...]
3  Ha N>M akkor
4      K:=N
5  Egyébként
6      K:=M
7  Feltétel vége
8  Csor[K]:=[]
9  db:=0
10 i:=1
11 Amíg i<=N addig
12     j:=1
13     Amíg j<=M és Asor[i] neq Bsor[j] addig
14         j:=j+1
15     Ciklus vége
16     Ha j<=M akkor
17         db:=db+1
18         Csor[db]:=Asor[i]
19     Feltétel vége
20     i:=i+1
21 Ciklus vége
```

21. ábra. Két sorozat közös elemeinek kiválogatása

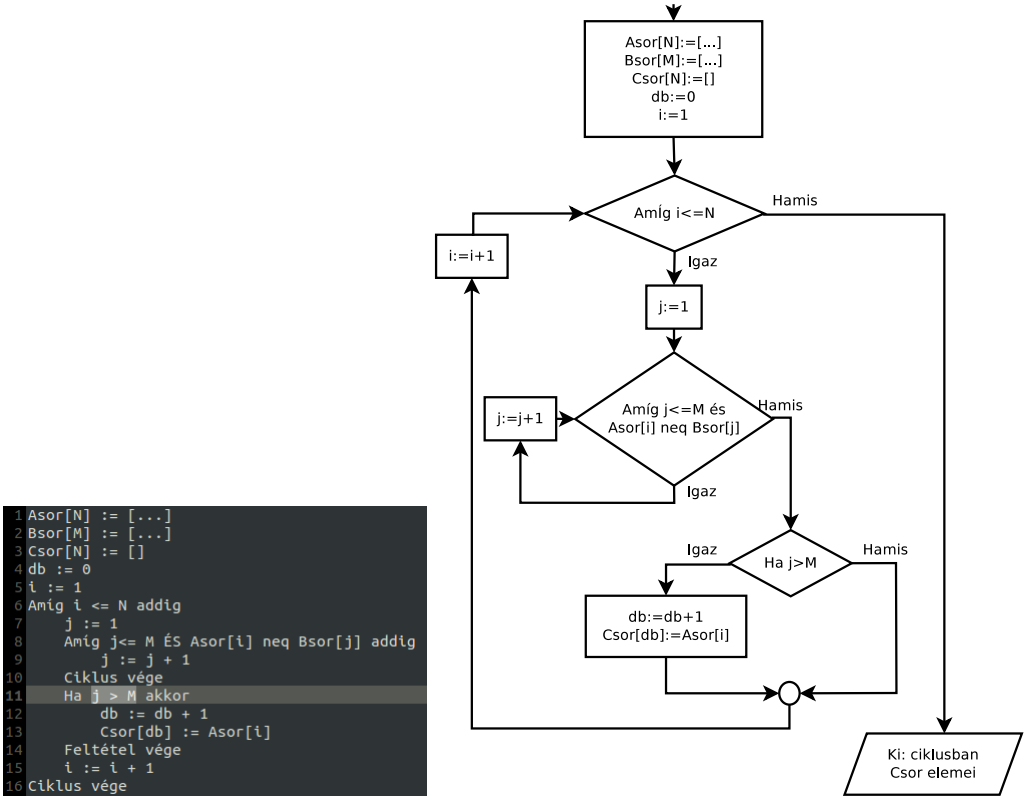
A megoldás során felhasználjuk a kiválogatás tételnél tanultakat: sorra vesszük az A sorozat elemeit és mindegyikről eldöntjük, hogy szerepel-e B-ben. Ha igen, kap egy sorszámot és betesszük az elemet a C sorozatba. A C sorozat maximális elemszáma N és M közül a kisebbik.

7.1.1. Feladatok

1. Adott a páros számok halmaza 100-ig és a négyetszámok halmaza 100-ig. Válogassuk ki a páros (ÉS) négyzetszámokat.
2. Adott a prímszámok halmaza 1000-ig és a kétjegyű számok halmaza. Válogassuk ki a kétjegyű (ÉS) prímszámokat.
3. Adott az egyjegyű számok halmaza és a négyzetszámok halmaza 100-ig. Melyek azok az egyjegyű számok, amelyeknek a négyzete is egyjegyű?

7.2. Különbség

Adott egy N elemű A sorozat, és egy M elemű B sorozat. A feladat az A sorozat azon elemeinek kiválogatása egy C sorozatba, amelyek nem szerepelnek B-ben ($A \setminus B$). C maximális elemszáma N. Ez az algoritmus is a kiválogatás tétel egyik alkalmazása. A feladat megoldása során sorra vesszük az A sorozat elemeit és mindegyikről eldöntjük, hogy szerepel-e B-ben. Ha nem, kap egy sorszámot és betesszük az elemet a C sorozatba. A metszetképzés algoritmusától egy ponton tér el: az elem akkor kerül a C sorozatba, ha *nem* szerepel B-ben.




```
1  Asor[N] := [...]
2  Bsor[M] := [...]
3  Csor[N] := []
4  db := 0
5  i := 1
6  Amíg i <= N addig
7      j := 1
8      Amíg j<= M ÉS Asor[i] neq Bsor[j] addig
9          j := j + 1
10     Ciklus vége
11     Ha j > M akkor
12         db := db + 1
13         Csor[db] := Asor[i]
14     Feltétel vége
15     i := i + 1
16 Ciklus vége
```

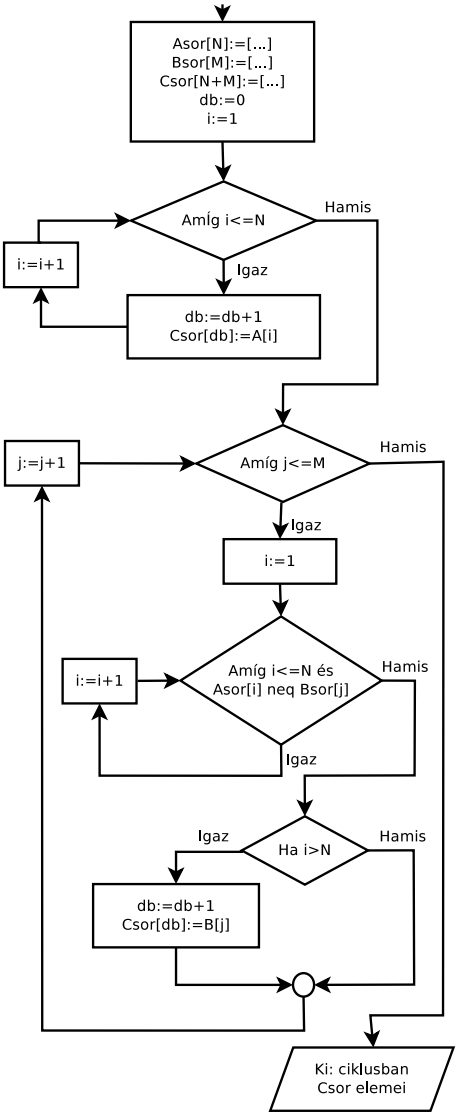
22. ábra. Az A sorozat azon elemei, amik nincsenek B-ben

7.2.1. Feladatok

1. Adott a páros számok sorozata 100-ig és a négyzetszámok sorozata 100-ig. Válogassuk ki a NEM páros négyzetszámokat.
2. Adott a prímszámok halmaza 1000-ig és a kétjegyű számok halmaza. Válogassuk ki a NEM kétjegyű prímszámokat.
3. Adott a prímszámok halmaza 1000-ig és a kétjegyű számok halmaza. Válogassuk ki a kétjegyű számok közül azokat, amelyek NEM prímek.

7.3. Egyesítés (unióképzés)

Adott egy N elemű A sorozat, és egy M elemű B sorozat. A feladat a két sorozat összes elemének kigyűjtése (ismétlődés nélkül) egy harmadik C sorozatba, ahol C maximális elemszáma $N + M$.




```
1  Asor[N] := [...]
2  Bsor[M] := [...]
3  Csor[N+M] := []
4  db := 0
5  i := 1
6  Amíg i <= N addig
7      db := db + 1
8      Csor[db] := Asor[i]
9      i := i + 1
10 Ciklus vége
11 Amíg j <= M addig
12     i := 1
13     Amíg i <= N ÉS Asor[i] neq Bsor[j] addig
14         i := i + 1
15     Ciklus vége
16     Ha i > N akkor
17         db := db + 1
18         Csor[db] := Bsor[j]
19     Feltétel vége
20     j := j + 1
21 Ciklus vége
```

23. ábra. Az A és a B sorozat egyesítése

Első lépésben az A sorozat összes elemét beletesszük a C -be. Majd sorra vesszük a B sorozat elemeit és mindegyikről eldöntjük, hogy szerepel-e A-ban. Ha nem, kap egy sorszámot és betesszük a C-be. A második lépéstől (ciklustól) kezdve megegyezik a különbségképzés algoritmusával ($B \setminus A$).

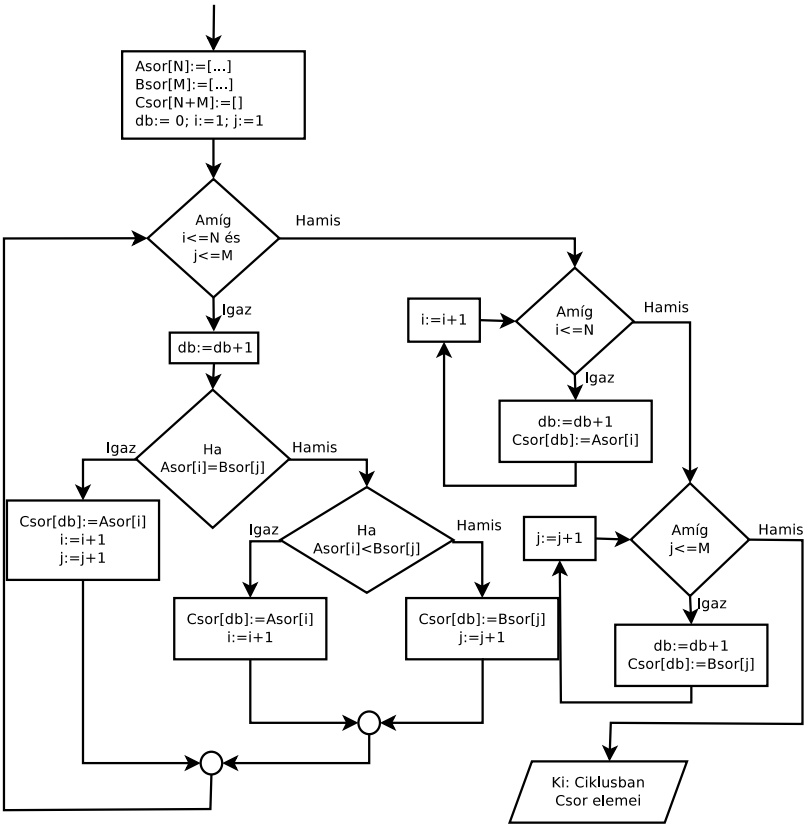
7.3.1. Feladatok

1. Adott 100-ig a 3-mal osztható számok sorozata, és az 5-tel osztható számok sorozata. 1 és 100 között mely számok oszthatók 3-mal VAGY 5-tel.
2. Egy általános iskolában adott a fiúk magasságát, illetve külön a lányok magasságát tartalmazó sorozat. Kosárlabda edzésre a 140 cm-nél magasabb fiúk VAGY lányok jelentkezhetnek. Válogassuk ki őket!
3. Földrészenként külön sorozatban tároljuk az országok népességeit. Válogassuk ki az európai VAGY ázsiai országok közül az 50 millió főnél népesebb országokat.

7.4. Összefuttatás

Az egyesítés speciális esete. Adott egy N elemű rendezett A sorozat, és egy M elemű rendezett B sorozat. A feladat a két rendezett sorozat egyesítése úgy, hogy a rendezettség megmaradjon. Az algoritmus kimenete tehát egy rendezett, az A és B sorozat összes elemét ismétlődés nélkül tartalmazó C sorozat, amelynek maximális elemszáma $N + M$.

```
Asor[N] := [...]
Bsor[M] := [...]
Csor[N+M] := []
db := 0
i := 1
j := 1
Amig i <= N ÉS j <= M addig
        db := db + 1
        Ha Asor[i] == Bsor[j] akkor
                Csor[db] := Asor[i]
                i := i + 1
                j := j + 1
        Egyébként
                Ha Asor[i] < Bsor[j] akkor
                        Csor[db] := Asor[i]
                        i := i + 1
                Egyébként
                        Csor[db] := Bsor[j]
                        j := j + 1
                Feltétel vége
        Feltétel vége
Ciklus vége
Amig i <= N addig
        db := db + 1
        Csor[db] := Asor[i]
        i := i + 1
Ciklus vége
Amig j <= M addig
        db := db + 1
        Csor[db] := Bsor[j]
        j := j + 1
Ciklus vége
```



24. ábra. Az A és a B rendezett sorozatok összefuttatása

7.4.1. Feladatok

1. A Miskolci Egyetem GÉIK karán a mérnök-, a gazdaság- és a programtervező informatikus alapszakokra külön-külön adott a felvettek rangsora 2014-re. Állítsuk elő a három informatikus alapszakra felvételt nyert hallgatók összesített rangsorát. *Megoldás:* először két sorozatot futtassunk össze, majd futtassuk össze ennek a műveletnek az eredményét a harmadik sorozattal.
2. Földrészenként külön sorozatban tároljuk a folyók hosszát csökkenő sorrendben. Állítsuk elő az európai vagy ázsiai folyók összesített csökkenő listáját. *Figyelem!* A 24. ábrán látható algoritmus növekvő sorozatot készít.

Hivatkozások

[1] Gregorics Tibor, Heizlerné Bakonyi Viktória, Horváth Győző, Menyhárt László, Pap Gábor Sándorné, Papp-Varga Zsuzsanna, Szlávi Péter, and Zsakó László. *Programozási alapismeretek*. Digitális Tankönyvtár, 2012.
[2] Simon Gyula. *A programozás alapjai*. Typotex Kiadó, Digitális Tankönyvtár, 2011.
[3] Dr. Szepesné Stiftinger Mária. *Informatika 3. – Programozási ismeretek*. Digitális Tankönyvtár, 2010.
[4] Mester Gyula. *Adatstruktúrák és algoritmusok példatár*. Typotex Kiadó, Digitális Tankönyvtár, 2011.
[5] Marton László and Fehérvári Arnold. *Algoritmusok és adatstruktúrák*. SZIF-UNIVERSITAS Kft, 2001.
[6] Geda Gábor and Hernyák Zoltán. *Algoritmizálás és adatmodellek*. Digitális Tankönyvtár, 2011.
[7] Dr. Nyakóné dr. Juhász Katalin, Dr. Terdik György, Biró Piroska, and Dr. Kátai Zoltán. *Bevezetés az Informatikába*. Digitális Tankönyvtár, 2011.
[8] O.J. Dahl, E.W. Dijkstra, and C.A.R. Hoare. *Structured Programming*. Academic Press, London, 1972.
[9] Juhász István, Kósa Márk, and Pánovics János. *C példatár*. Panem Kft., Digitális Tankönyvtár.
[10] Házy Attila and Nagy Ferenc. *Adatstruktúrák és algoritmusok*. Digitális Tankönyvtár, 2011.
[11] Aszalós László and Herendi Tamás. *Algoritmusok*. Digitális Tankönyvtár, 2011.
[12] Tömösközi Péter. *Algoritmizálás alapjai*. Digitális Tankönyvtár, 2011.

Köszönetnyilvánítás

Az oktatási segédlet kidolgozása a Miskolci Egyetem társadalmi – gazdasági szerepének fejlesztése, különös tekintettel a duális képzési típusú megoldásokra témakörű K+F projektje keretében - TÁMOP-4.1.1.F-13/1-2013-0010 - az „ELTÉRŐ UTAK A SIKERES ÉLETHEZ" projekt részeként – az Új Széchenyi Terv keretében – az Európai Unió támogatásával, az Európai Szociális Alap társfinanszírozásával valósult meg.