

Elemi algoritmusok

Számjegyek és oszthatóság

dr. Ionescu Klára

clara@cs.ubbcluj.ro

Mit nevezünk elemi algoritmusnak?

- Egyszerű „recept” egy adott feladat megoldására, ahol:
 - Nincs szükség összetett adatszerkezetre
 - Aritmetikai, logikai és relációs műveleteket alkalmazunk
 - Az eredmény könnyen generálható, értelmezhető
- Ugyanakkor: az illető feladat megjelenhet más feladat részfeladataként
- Általános megoldást adunk, ahhoz, hogy újra-felhasználható legyen

Alapszabályok

- Minden feladatot részfeladatokra bontunk, a részfeladatok megoldásait alprogramok (al-algoritmusok) formájában adjuk meg
- Egy alprogram = egy funkció
- Egy alprogram nem olvas be, nem ír ki, csak ha pontosan ez a funkciója
- A programegységek a paramétereik segítségével kommunikálnak egymással
- Minden alprogramnak pontosan annyi paramétere lesz, ahányra szüksége van (minden, amit kintről kap, és minden, amit a hívás helyére továbbítania kell)

1. Kedvcsináló érdekes feladat: Elnökválasztás

Egy elnökválasztás alkalmával igen sok jelölt indul. Nem tudjuk hány, de biztos több, mint 1 millió. Minden elnökjelölthöz egy természetes számot rendeltek. Állapítsuk meg a győztest, ha van ilyen!

Másként:

- Adva van nagyon sok nagy szám
- Ezeket nem lehet egy tömbben tárolni
- Meg kell találnunk, (ha létezik!), azt a számot, amely legalább a számok fele plusz 1-szer megtalálható az adott számok között

Példák

$N = 10$

1, 2, 1, 1, 1, 2, 2, 2, 1, 1

Nyerő: 1

1, 2, 1, 2, 1, 2, 1, 2, 1, 2

Nem nyer senki

1, 2, 1, 2, 1, 2, 1, 2, 1, 1

Nyerő: 1

1, 1, 1, 1, 2, 3, 4, 5, 1, 1

Nyerő: 1

1, 2, 3, 4, 5, 6, 1, 1, 1, 1

Nem nyer senki

Elemzés: mit lehet tenni egy számmal, ahhoz, hogy maximális értékű információt facsarjunk belőle?

Algoritmus Szavazás:

Be: szám

jelölt \leftarrow szám

hány \leftarrow 1

Amíg *nincs vége az állománynak* **végezd el:**

Be: szám

Ha szám = jelölt **akkor**

 hány \leftarrow hány + 1

különb

Ha hány = 0 **akkor**

 jelölt \leftarrow szám

 hány \leftarrow 1

különb

 hány \leftarrow hány - 1

vége(ha)

vége(ha)

vége(amíg)

$\text{hány} \leftarrow 0$

$n \leftarrow 0$

Amíg *nincs vége az állománynak* **végezd el:**

Be: szám

$n \leftarrow n + 1$

Ha szám = jelölt **akkor**

$\text{hány} \leftarrow \text{hány} + 1$

vége(ha)

vége(amíg)

Ha $\text{hány} > n \text{ div } 2$ **akkor**

Ki: 'Nyertes a ',jelölt,','

különben

Ki: 'Sajnos nem nyert senki!'

vége(ha)

Vége(algoritmus)

2. Két természetes szám legnagyobb közös osztója

Határozzuk meg két természetes szám legnagyobb közös osztóját!

Megoldás

Legyen $a, b \in \mathbb{N}^*$ és a következő jelölések:

(a, b) : a és b legnagyobb közös osztója;

Algoritmus Lnko(a, b, lnko):

{ Funkció: meghatározza a és b legnagyobb közös osztóját }

{ Bemeneti paraméterek: a, b }

{ Kimeneti paraméter: lnko }

{ ismételt kivonásokkal }

Amíg $a \neq b$ **végezd el:**

Ha $a > b$ **akkor**

$a \leftarrow a - b$

különben

$b \leftarrow b - a$

vége(ha)

vége(amíg)

$\text{lnko} \leftarrow a$ *{ kimeneti paraméter nélkül: „térítsd a” }*

Vége(algoritmus)

Kivonások helyett osztásokkal

Algoritmus L_{nko}(a, b, l_{nko}):

{ *Funkció: meghatározza a és b legnagyobb közös osztóját* }

{ *Bemeneti paraméterek: a, b* }

{ *Kimeneti paraméter: l_{nko}* }

Ismételd

{ *ismételt osztásokkal* }

maradék \leftarrow a **mod** b

a \leftarrow b

b \leftarrow maradék

ameddig maradék \neq 0 { *kilépünk, amikor maradék = 0* }

l_{nko} \leftarrow a { *kimeneti paraméter nélkül: „térítsd a”* }

Vége(algoritmus)

3. Mi a hatása?

Algoritmus $\text{Mi_ez}(a, b, \text{érték})$:

Amíg $a \bmod b \neq 0$ **végezd el**:

$a \leftarrow b$

$b \leftarrow a \bmod b$

vége(amíg)

$\text{érték} \leftarrow b$

Vége(algoritmus)

Úgy néz ki, mintha Euklidész algoritmus a lenne, de vegyük észre, hogy, mivel nincs segédváltozó, b kiszámolásakor a-nak megváltoztatott értékét használjuk. Ezért, bármely a és b értékek esetében b 0 lesz az első lépésben, így az Amíg feltételében már 0-val osztunk. Tehát a hatás: hibaüzenet (Nullával osztás).

4. Fordított szám

„Fordítsunk” meg adott (legtöbb 9 számjegyű) természetes számot! (Generáljuk az adott szám számjegyeit az eredetivel fordított sorrendben tartalmazó számot!)

Elemzés

Az **újszám** kezdeti értéke nulla, majd minden lépésben meghatározzuk **újszám** új értékét a *Horner* sémával.

Algoritmus fordítottSzám(szám, újSzám):

{ *bemeneti adat: szám, kimeneti adat: újSzám* }

újSzám \leftarrow 0

Amíg szám \neq 0 **végezd el:**

újSzám \leftarrow újSzám * 10 + szám **mod** 10 { *Horner séma* }

szám \leftarrow szám **div** 10

vége(amíg)

Vége(algoritmus)

5. Palindromszám

Adott természetes számról döntsük el, hogy palindromszám-e vagy sem! Egy számot *palindromszámnak* (vagy *tükörszámnak*) nevezünk, ha egyenlő a „fordított”-jával, vagyis azzal a számmal, amelyet a szám számjegyei fordított sorrendben alkotnak.

Megoldás

- A számot számjegyekre bontjuk.
- Hasonlóan az előző algoritmushoz, ezzel párhuzamosan felépítjük az új számot.
- Az új szám generálását ugyancsak a **Horner-séma** néven ismert módszer segítségével végezzük: $\text{újszám} \leftarrow \text{újszám} \cdot 10 + \text{szj}$.

5. Palindromszám

- Az algoritmus a számjegyeket úgy határozza meg, hogy ismételten osztja az eredeti számot 10-zel

⇒ az algoritmus végén az eredeti szám értéke 0.

De: nekünk szükségünk van az eredeti értékre ahhoz, hogy összehasonlíthassuk az új számmal!!!

- Ezért a beolvasott számról a feldolgozás előtt *másolatot* készítünk.

Algoritmus Palindrom(szám, p):

{ bemeneti adat: szám, kimeneti adat: p }

{ p = igaz, ha szám palindrom, különben hamis }

másolat \leftarrow szám

újszám \leftarrow 0

Amíg szám > 0 **végezd el:**

számjegy \leftarrow szám **mod** 10 *{ számjegyekre bontás }*

újszám \leftarrow újszám*10 + szj *{ Horner séma }*

szám \leftarrow szám **div** 10 *{ a feldolgozott számjegyet levágjuk }*

vége(amíg)

{ ugyanaz mint:

p \leftarrow újszám = másolat

Ha újszám = másolat **akkor**

p \leftarrow igaz

különben

p \leftarrow hamis

vége(ha) *de jobb!!! }*

Vége(algoritmus)

6. Törzstényezők

Bontsuk törzstényezőkre az adott n számot!

Elemzés

- A számot ismételten osztjuk, előbb 2-vel, majd 3-mal stb. ameddig 1-gyé nem válik.
- Minden osztóval addig osztunk amíg a maradék 0.
- Amint egy bizonyos osztó már nem osztja maradéktalanul a számunkat, kiírjuk – az osztások számát hatványként használva.
- A kiírást figyelmesen végezzük!
- $2\ 2\ 3 \rightarrow 1p$
- $2^2 * 3 \rightarrow 2p$

Példa

12	2
6	2
3	3
1	

Algoritmus Törzstényezők(n):

osztó $\rightarrow 2$

{ *Bemeneti adat: n* }

Ismételd

hatvány $\rightarrow 0$

Amíg $n \bmod \text{osztó} = 0$ **végezd el:**

hatvány $\rightarrow \text{hatvány} + 1$

$n \rightarrow n \text{ div osztó}$

vége(amíg)

Ha hatvány $\neq 0$ **akkor**

Ki: osztó, '^', hatvány, ' '

Ha $n \neq 1$ **akkor** **Ki:** '*'

vége(ha)

vége(ha)

Ha osztó = 2 **akkor** osztó \rightarrow osztó + 1

különben osztó \rightarrow osztó + 1

ameddig $n = 1$

Vége(algoritmus)

7. Tökéletes szám

Keressük meg és írjuk ki az első n ($n \leq 4$) tökéletes számot! Egy szám tökéletes, ha egyenlő a nála kisebb osztóinak összegével.

Példa: $6 = 1 + 2 + 3$.

Elemzés:

- Az első szám amit vizsgálunk: 2
- Osztókat csak a szám feléig érdemes keresni
- Amint találunk egy osztót, hozzáadjuk egy segédváltozó értékéhez
- Ha ez a szám egyenlő a vizsgált számmal, találtunk egy tökéletes számot.

Algoritmus TökéletesSzámokKiírása(hány):

{ Bemeneti adat: hány, a kiírandó tökéletes számok száma }

darab \leftarrow 0; szám \leftarrow 2 *{ Még nincs egy tökéletes szám sem }*

Amíg darab < hány **végezd el:**

osztókÖsszege \leftarrow 1 *{ Az első osztó az 1 }*

ngy \leftarrow négyzetgyök(szám) *{ elkerülhető ngy kiszámítása?? }*

Minden osztó = 2, ngy **végezd el:**

Ha szám **mod** osztó = 0 **akkor**

osztókÖsszege \leftarrow osztókÖsszege + osztó *{ osztók összege }*

Ha osztó \neq szám **div** osztó **akkor**

{ ha a szám nem négyzetszám }

osztókÖsszege \leftarrow osztókÖsszege + szám **div** osztó

vége(ha)

vége(ha)

vége(minden)

Ha osztókÖsszege = szám **akkor**

Ki: szám

darab \leftarrow darab + 1

vége(ha)

szám \leftarrow szám + 1 { *A következő vizsgálandó szám* }

vége(amíg)

Vége(algoritmus)

Lépések finomítása

- Bonyolultabb feladatok esetében az algoritmus leírása nem könnyű feladat. Ezért célszerű először a megoldást körvonalazni, és csak azután részletezni:

A **feladat elemzése** során sor kerül:

- a **bemeneti és kimeneti adatok** megállapítására;
- a megfelelő **adatszerkezetek** kiválasztására és megtervezésére;
- a feladat **követelményeinek** szétválasztására;
- a megoldási **módszer** megállapítására;
- a megoldás lépéseinek leírására.

Lépések finomítása

Lépések finomítása: folyamat, amely tart az algoritmus **kezdeti vázlatától**, a **végleges, kidolgozott algoritmusig**.

- Kiindulunk a feladat **specifikációból** és **fentről lefele** megtervezzük az algoritmust.
- Újabb meg újabb **változatok**at dolgozunk ki, amelyek eleinte tartalmaznak magyarul leírt magyarázó sorokat, majd ezeket standard utasításokra írjuk át.
- Így az algoritmus változatai mindegyre bővülnek egyik változattól a másikig.

Végül: kódolás, majd tesztelés

8. Fibonacci-számok

Generáljuk és írjuk ki az első n Fibonacci-számot!

Elemzés

- **Bemeneti adatok:** n természetes szám
- **Kimeneti adatok:** n darab Fibonacci-szám
- nincs szükség adatszerkezetre (a feladat nem kéri ezek megőrzését sorozat formájában)
- a feladat követelményeinek szétválasztása:
 - beolvasunk egy természetes számot,
 - kiszámítunk és kiírunk sorban egy-egy Fibonacci-számot

Megoldási módszer

- A *Fibonacci-számok*at az alábbi rekurzív összefüggéssel definiáljuk:

$$F_0 = 0, F_1 = 1, F_i = F_{i-1} + F_{i-2}, \text{ ha } i \geq 2.$$

- Minden *Fibonacci-szám* tehát a megelőző kettőnek az összege.
- A *Fibonacci-számok* kapcsolatban vannak az *aranymetszés* arányával.
- A *Fibonacci-számok* exponenciálisan nőnek.
- A *Fibonacci-számok* sorozata:

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, ...

A megoldás lépései

Algoritmus Fibonacci(n):

Kezdőértékekadások: a, b, c

Ki: a, b, c

$k \leftarrow 3$

Amíg $k < n$ **végezd el:**

egy Fibonacci szám kiszámítása c-ben

Ki: c

$k \leftarrow k + 1$

Vége(amíg)

Vége(algoritmus)

Lépések finomítása

- Az első n Fibonacci szám generálásához elegendő három változó: a , b és c .
- A számsorozat egy új tagját (c) úgy állítjuk elő, hogy a már kiszámolt elemeket *balra „toljuk”* egy pozícióval.
- Így rendre megkapjuk a és b új értékét, majd ezek ismeretében kiszámíthatjuk c új értékét.
- Ezeket a lépéseket addig ismételjük, amíg a kért számú elemet elő nem állítottuk!
- A fenti vázlat módosítása: ha $n = 1$ vagy $n = 2$, csak egy, illetve két számot kell kiírnunk.
- A kidolgozásban leírjuk c új értékének kiszámítását

Algorithmus Fibonacci(n):

$a \leftarrow 0$

b ← 1 { *Bemeneti adat: n* }

Ha $n = 1$ akkor

Ki: a

különben

Ha $n = 2$ akkor

Ki: a, b

különben

$c \leftarrow a + b$

Ki: a, b, c

$$k \leftarrow 3$$

Amíg $k < n$ végezd el:

$$a \leftarrow b$$
$$b \leftarrow c$$

c ← a + b

Ki: c

$$k \leftarrow k + 1$$

vége(amíg)

vége(ha)

vége(ha)

Vége(algoritmus)

9. Fibonacci-szám-e?

Állapítsuk meg egy adott ($n > 1$) számról, hogy Fibonacci-szám-e?
Ha nem, bontsuk fel különböző (minimális számú) Fibonacci-számok összegére.

Elemzés

Az előbbi algoritmussal generáljuk a *Fibonacci*-számokat, amíg egy olyan számot nem kapunk, amely nagyobb mint az adott szám vagy egyenlő az adott számmal.

- Ha c utolsó értéke *Fibonacci* szám, az algoritmus véget ér.
- Különben kiírjuk azt a legnagyobb *Fibonacci* számot, amely kisebb vagy egyenlő az adott számmal (ez a b -ben található).
- Majd b -t kivonjuk n aktuális értékéből.
- Ha b egyenlő n aktuális értékével, az algoritmus véget ér.
- Különben megismételjük az ellenőrzést.

Fibonacci-szám-e?

- Amíg n (amelyből minden lépésben kivonjuk a b Fibonacci-számot, amelyet kiírnunk) pozitív, meghatározunk egy-egy újabb b Fibonacci-számot, amely kisebb vagy egyenlő az adott szám aktuális értékével.
- *A feladat egyedi felbontást kér.* Ezt biztosítja az elemek növekvő sorrendben való megkeresése.
- Vegyük észre, hogy ez a megkötés nem teszi lehetővé, hogy az összegként felírt számban szerepeljen két egymás utáni eleme a *Fibonacci* sorozatnak.

Algoritmus Fibonacci_szám_e(n): { az n számot vizsgáljuk }
 $a \leftarrow 0$; $b \leftarrow 1$; $c \leftarrow a + b$
 Amíg $c < n$ **végezd el:**
 $a \leftarrow b$
 $b \leftarrow c$
 $c \leftarrow a + b$ { b a legnagyobb Fibonacci szám, amely $< n$ }
 vége(amíg) // $c \geq n$
 Ha $c = n$ **akkor** **Ki:** 'Fibonacci szám'
 különben **Ki:** $n, '=', b$; $n \leftarrow n - b$
 Amíg $n > 0$ **végezd el:** { amíg van még maradék az adott számból }
 Amíg $b > n$ **végezd el:** { elindulunk visszafele }
 $c \leftarrow b$
 $b \leftarrow a$
 $a \leftarrow c - b$ { b a legnagyobb Fibonacci szám, amely $\leq n$ }
 vége(amíg)
 Ki: $+', b$; $n \leftarrow n - b$
 vége(amíg)
 vége(ha)
Vége(algoritmus)

10. Kettőhatvány

Adott az n nem nulla természetes szám ($0 < n \leq 10^9$). Döntsük el, hogy kettőhatvány-e! Ha az adott szám nem kettőhatvány, bontsuk kettőhatványok összegére!

Elemzés

- Generáljuk a kettőhatványokat, amíg kisebbek, mint n
- Miután leállt a generálás, megvizsgáljuk, hogy a legutolsó kettőhatvány egyenlő-e n -nel és így eldöntjük, a választ az kérdésre
- Ha az utolsó kettőhatvány nagyobb, mint n , osztjuk kettővel, így megvan a felbontás első tagja
- Ezt kivonjuk n -ből

Kettőhatvány

- A következő tagokat megkaphatnánk, ha újra végrehajtanánk az előbb leírt műveletsort, amíg n nullává nem válik
- Észrevétel: így újból előlről kezdenénk a kettőhatványok generálását; ha visszafele haladunk, osztva kettővel az aktuális kettőhatványt, hatékonyabb megoldáshoz jutunk
- Ugyanakkor, ha nem lenne a második követelmény, a döntést sokkal egyszerűbben így oldhatnánk meg:

Algoritmus kettőHatvány(n):

térítsd $n \ \& \ (n - 1) = 0$

Vége(algoritmus)

Algoritmus kettőHatvány(n, m):

// m kimeneti paraméter

$m \leftarrow 1$

Amíg $m < n$ **végezd el:**

$m \leftarrow m * 2$

vége(amíg)

térítsd $m = n$

Vége(algoritmus)

Algoritmus felbontás(n, m, k, tagok): // ha k értéke 0 marad \Rightarrow n kettőhatvány

$k \leftarrow 0$

Ha nem kettőHatvány(n, m) **akkor**

Amíg $n > 0$ **végezd el:**

Amíg $m > n$ **végezd el:**

$m \leftarrow m / 2$

vége(amíg)

$k \leftarrow k + 1$

$\text{tagok}[k] \leftarrow m$

$n \leftarrow n - m$

vége(amíg)

vége(ha)

Vége(algoritmus)

11. Mi a hatása?

Algoritmus $Mi_ez(n)$:

$a = 1$

$b = 0$

Minden $k = 1, n$ **végezd el:**

Ki: b

$b = a + b$

$a = b - a$

vége(minden)

Vége(algoritmus)

Algoritmusok optimalizálása

- Optimalizáláskor egy **kész** algoritmus **hatékonyságát próbáljuk növelni**.
- Tehát túl vagyunk a **finomításon**.
- Az algoritmus teljesen kész, de elégedetlenek vagyunk a **teljesítménnyel**.
- Ha az algoritmus egyetlen **Minden** ciklust tartalmaz, megvizsgáljuk, hogy valóban minden adatot föl kell dolgoznunk?
- Vajon nem állítható le a feldolgozás hamarabb?
- Néha észrevesszük, hogy esetleg létezik egy **$O(\log n)$** bonyolultságú algoritmus...

Algoritmusok optimalizálása

- Ha egy algoritmus bonyolultsága *lineáris*, keresünk egy logaritmikusát (lásd **Gyorshatvány**), ha nem sikerül, megpróbáljuk csökkenteni a lépések számát!
- Csökkentjük a bonyolultságfüggvényt:
 1. Megpróbáljuk gyorsítani az algoritmust **vagy**
 2. Kevesebb memóriaigénnyel akarjuk megoldani a problémát
- *Igazi optimalizálás akkor történik, ha anélkül sikerül jobb algoritmust találni, hogy változna a másik bonyolultság, vagy szerencsés esetben az is csökken.*

12. Gyorshatvány

Adottak az $x > 0$ valós és az $n > 6$ természetes számok. Számítsuk ki az x^n számot minél kevesebb szorzással!

Elemzés

A feladat triviális megoldása $n - 1$ szorzással történne (**Minden** típusú struktúrával):

...

eredmény \leftarrow alap

Minden $i=2, n$ **végezd el:**

eredmény \leftarrow eredmény * alap

vége(minden)

...

Gyorshatvány

- A feladat megoldható kevesebb szorzással is, ha a **kitevőt** kifejezzük a **2**-es számrendszerben, és x^n -t felírjuk x^{2^k} alakú számok szorzataként ($k > 0$).
- Az x^{2^k} számokat kiszámolhatjuk egymást követő négyzetre emelésekkel.

Példa

Mivel $9 = (1001)_2$,

$$x^9 = x^8 \cdot x = ((x^2)^2)^2 \cdot x$$

Ennek alapján a következő algoritmust már **Abu-Ja far Mohammed ibn Mura al-Kvarîzmi** (780 k.–850 k.) arab matematikus ismerte:

Algoritmus Gyorshatvány(alap, kitevő, eredmény):

{ Bemenet: alap, kitevő. Kimenet: eredmény }

eredmény \leftarrow 1

Amíg kitevő > 0 **végezd el:**

Ha kitevő *páratlan akkor*

eredmény \leftarrow eredmény * alap

vége(ha)

alap \leftarrow alap * alap

kitevő \leftarrow kitevő **div** 2

vége(amíg)

Vége(algoritmus)

13. Példa: Orosz szorzás

Legyen $a, b \in \mathbf{N}^$. Számítsuk ki a és b szorzatát! Tudjuk, hogy az orosz muzsik csak a következő műveleteket tudja elvégezni:*

- el tudja dönteni, hogy egy szám páros vagy páratlan;
- össze tud adni két számot;
- össze tud hasonlítani egy számot 0-val;
- felezni tud egy számot (elosztani 2-vel).

Példa

x	45	45	22	11	5	2	1
y	17	17	34	68	136	272	544
p	0	17		85	221		765

Algoritmus Orosz_szorzás(a, b, p):

{ Kiszámítjuk a és b szorzatát, „szorzás” nélkül }

{ Bemeneti adatok: a, b }

{ Kimeneti adat: p (a szorzat) }

$p \leftarrow 0$

Amíg $x > 0$ **végezd el:**

Ha x páratlan **akkor**

$p \leftarrow p + y$

vége(ha)

$x \leftarrow x \text{ div } 2$

$y \leftarrow y + y$

vége(amíg)

Vége(algoritmus)

14. Prímszám-e?

Döntsük el az n számról, hogy prímszám-e! (Létezik-e egész osztója önmagán és az 1-en kívül?)

Elemzés:

- A prímszám definíciójából indulunk ki: **egy szám akkor prím, ha pontosan két osztója van: 1 és maga a szám**
- **Első ötlet:** az algoritmus számolja meg az adott szám osztóit, elosztva ezt sorban valamennyi számmal **1**-től n -ig
- A döntésnek megfelelő üzenetet az osztók száma alapján írjuk ki

Prím-e?

Algoritmus Prím_1(n, prím): *{ naiv algoritmus!!! }*
 osztók_száma \leftarrow 0
 Minden osztó=1, n **végezd el**:
 Ha n **mod** osztó = 0 **akkor**
 osztók_száma \leftarrow osztók_száma + 1
 vége(ha)
 vége(minden)
 prím \leftarrow osztók_száma = 2
 { értéket adtunk a prím változónak }
 *{ C++-ban: **return** osztók_száma == 2 }*
Vége(algoritmus)

Optimalizálások

Észrevétel: az osztások száma fölöslegesen nagy.

- Ezt a számot csökkenteni lehet, mivel ha **2** és **$n/2$** között nincs egyetlen osztó sem, akkor biztos nincs **$n/2$** és **n** között sem.
- Sőt, elég a szám **négyzetgyökéig** keresni a lehetséges osztót.

Algoritmus Prím_2(n , prím):

$\text{prím} \leftarrow \text{igaz}$

$\text{ngy} \leftarrow \text{négyzetgyök}(n)$

Minden osztó=2, ngy **végezd el:**

Ha $n \bmod \text{osztó} = 0$ **akkor**

$\text{prím} \leftarrow \text{hamis}$

vége(ha)

vége(minden)

Vége(algoritmus)

További optimalizálások

Észrevétel: a logikai változónak lehet több munkát adni.

Leállítjuk a munkát abban a pillanatban, amikor találtunk egy osztót és a **prím** logikai változó hamissá vált.

- Lemondunk a **Minden** típusú ciklusról és egy **Amíg** vagy **Ismételd** típusú ciklussal helyettesítjük.
- Mivel **n** nem változik a ciklusmagban, a négyzetgyököt csak egyszer számítjuk ki, a ciklusba lépés előtt.

Algoritmus Prím_3(n , **prím**):

prím \leftarrow igaz

osztó \leftarrow 2

ngy \leftarrow négyzetgyök(n)

Amíg **prím és** ($\text{osztó} \leq \text{ngy}$) **v.e.:**

Ha $n \bmod \text{osztó} = 0$ **akkor**

prím \leftarrow hamis

különb

osztó \leftarrow **osztó** + 1

vége(ha)

vége(amíg)

Vége(algoritmus)

További optimalizálások

Észrevétel: *a páros számok mind oszthatók 2-vel, és a 2 kivételével nem prímek!*

- De ha „megszabadultunk” a páros számok fölösleges vizsgálatától, akkor fölösleges páros számokkal osztani, hiszen páratlan számnak csak páratlan osztói lesznek!
- Ahhoz, hogy az algoritmusunk tökéletesen működjön, akkor is, ha **$n = 1$** , a következőképpen járunk el:

Algoritmus Prím_4(n , prím):

Ha $n = 1$ **akkor**

$\text{prím} \leftarrow \text{hamis}$

különben

Ha n páros **akkor** $\text{prím} \leftarrow n = 2$

különben

$\text{prím} \leftarrow \text{igaz}; \quad \text{osztó} \leftarrow 3$

$\text{ngy} \leftarrow \text{négyzetgyök}(n)$

Amíg prím **és** $(\text{osztó} \leq \text{ngy})$ **végezd el:**

Ha $n \bmod \text{osztó} = 0$ **akkor**

$\text{prím} \leftarrow \text{hamis}$

különben

$\text{osztó} \leftarrow \text{osztó} + 2$

vége(ha)

vége(amíg)

vége(ha)

vége(ha)

Vége(algoritmus)

*{ tudjuk, hogy a prímszámok 6 többszöröseinél eggyel
kisebbek vagy nagyobbak }*

Algoritmus Prím_5(határ):

Ha $n = 1$ **akkor**

$\text{prím} \leftarrow \text{hamis}$

különb

Ha n páros **akkor**

$\text{prím} \leftarrow n = 2$

különb

Ha $n \leq 5$ **akkor**

$\text{prím} \leftarrow \text{igaz}$

különb

Ha $((n - 1) \bmod 6 \neq 0)$ és $((n + 1) \bmod 6 \neq 0)$

akkor $\text{prím} \leftarrow \text{hamis}$

különb

$\text{osztó} \leftarrow 3$

... { tovább ugyanaz, mint az előző algoritmusban }

További optimalizálás

- Továbbá, ismeretes, hogy a négyzetgyököt számoló függvény ismeretlen lépésszámban határozza meg az eredményt, amely valós szám. Ezt elkerülendő, lemondunk a négyzetgyök kiszámításáról és az Amíg feltételét a következőképpen írjuk:

Amíg prím és $(\text{osztó} * \text{osztó} \leq n)$ végezd el:

- Így, nem dolgozunk valós számokkal és nem számítjuk ki fölöslegesen a négyzetgyököt.

15. Prímszámok generálása

Generáljuk egy adott számnál kisebb összes prímszámot!

Megoldás

Generáljuk a vizsgálandó számokat és ezekre alkalmazzuk az előbbi algoritmust:

Algoritmus Prímek_1(határ):

Ki: 2

$n \leftarrow 3$

Amíg $n < \text{határ}$ **végezd el:**

$\text{prím} \leftarrow \text{igaz}$

$\text{osztó} \leftarrow 3$

$\text{ngy} \leftarrow \text{négyzetgyök}(n)$

Amíg prím **és** $\text{osztó} \leq \text{ngy}$ **végezd el:**

Ha $n \bmod \text{osztó} = 0$ **akkor**

$\text{prím} \leftarrow \text{hamis}$

különben $\text{osztó} \leftarrow \text{osztó} + 2$

vége(ha)

vége(amíg)

Ha prím **akkor** **Ki:** n

$n \leftarrow n + 2$

vége(amíg)

Vége(algoritmus)

16. Eratosthenész szitája

- *Eratosthenész felírt minden számot **2** és **3000** között egy papiruszra, megjegyezte a **2**-t, mint **prímszámot**, majd **kihúzott minden páros számot**.*
- Most megjegyezte az első számot, amelyet még nem húzott ki (ez a **3**-as) és **kihúzott minden 3-mal osztható számot**.
- Az algoritmusban a számokat tárolnunk kell, hogy megjegyezhesük, melyek azok, amelyeket kihúztunk, illetve melyek azok, amelyeket nem.

Algoritmus Prímek_3(határ, prím):

{ határ-nál kisebb számokat vizsgálunk }

Minden $i=2$, határ **végezd el**:

$\text{prím}_i \leftarrow \text{igaz}$ *{ még nincs kihúzva egy szám sem }*

vége(minden)

Minden $i=2$, $i * i \leq \text{határ}$ **végezd el**:

Ha prím_i **akkor** *{ ha i még nincs kihúzva }*

$k \leftarrow i * i$ *{ az első kihúzandó szám }*

Amíg $k \leq \text{határ}$ **végezd el**:

$\text{prím}_k \leftarrow \text{hamis}$ *{ kihúzzuk a k számot }*

$k \leftarrow k + i$ *{ a következő kihúzandó többszöröse i -nek }*

vége(amíg)

vége(ha)

vége(minden)

Vége(algoritmus)

Végül, a **prím** sorozat alapján kiírhatók (generálhatók) a prímszámok.

17. Mi a hatása a következő algoritmusnak?

Bemeneti adat: egy pozitív egész szám

Kimeneti adat: egy szám

1. Jelöljük a bemeneti számot N -nel
2. Legyen $X = 0$ és $Z = 1$
3. Növeljük X értékét eggyel
4. Adjuk hozzá Z értékéhez X kétszeresét
5. Növeljük Z értékét eggyel
6. Ha Z nem nagyobb mint N , akkor ismételjük meg az algoritmust a 3. lépéstől
7. Írjuk ki X értékét

Általánosságok

- A számjegyekkel foglalkozó feladatok sok más feladat részfeladataiként jelentkeznek.
- Gyakran szükségünk van az „előfordulások”, illetve a „gyakoriság” tömbre.
- Ezek használata sok esetben vezet jobb (optimális) megoldásokhoz.
- Az oszthatóság vizsgálata szintén gyakran jelenik meg.

18. Hasonló számok

*Legyen x és y két természetes szám, amelyeknek legtöbb 9 számjegyük van. Döntsük el, hogy a két szám **hasonló-e**. Két számot **hasonlónak** nevezünk, ha számjegyeik halmaza azonos: 2131 és 32211 hasonló mivel számjegyeik halmaza ugyanaz: $\{1,2,3\}$.*

- a) Két tömb használatával
- b) Egy tömb segítségével
- c) Tömbhasználat nélkül

Hasonló számok

Elemzés

a) a feladat megoldása két tömb használatával rendkívül egyszerű:

- meghatározzuk x és y előfordulási tömbjét;
- összehasonlítjuk a két tömböt, és eldöntjük, hogy a két tömb azonos-e.

b) A feladat megoldása egy tömb használatával szintén egyszerű:

- meghatározzuk x előfordulási tömbjét;
- meghatározzuk, rendre az y szám számjegyeit, és vizsgáljuk, hogy ez a számjegy előfordult-e az x számban; ha y minden számjegyét megtaláljuk, következik, hogy x és y azonos. De ezt fordítva is megvizsgáljuk.

c) Ha nem szabad tömböt használni, meg kell vizsgálnunk, hogy x minden számjegye előfordul-e y -ban, és fordítva

Algoritmus előfordul(x, előf):

// a)

// létrehozuk az x szám számjegyeinek előfordulástömbjét

Minden $i = 0, 9$ **végezd el:**

előf[i] \leftarrow hamis

vége(minden)

Amíg $x > 0$ **végezd el:**

előf[x mod 10] \leftarrow igaz

$x \leftarrow x / 10$

vége(amíg)

Vége(algoritmus)

Algoritmus azonos_1(a, b): // eldöntjük, hogy a és b azonosak-e

előfordul(a, előfA); előfordul(b, előfB)

$i \leftarrow 0$

Amíg $i \leq 9$ **és** előfA[i] = előfB[i] **végezd el:**

$i \leftarrow i + 1$

vége(amíg)

térítsd $i > 9$

Vége(algoritmus)

Algoritmus ab(a, b): // b)

// az a előfordulástömbjében vizsgáljuk a b számjegyeinek megfelelő elemeket

előfordul(a, előfA)

Amíg $b > 0$ **és** előfA[b mod 10] **végezd el:**

$b \leftarrow b / 10$

vége(amíg)

térítsd $b = 0$

Vége(algoritmus)

Algoritmus azonos_2(x, y):

// x számjegyeit keressük y számjegyei között és fordítva

térítsd ab(x, y) **és** ab(y, x)

Vége(algoritmus)

Algoritmus AszámjegyeiBben(a, b): // c)
// vizsgáljuk, hogy a minden számjegye előfordul-e b-ben
 auxB \leftarrow b *// másolat b-ről*
Amíg a > 0 **végezd el:**
 aUtolsóSzej \leftarrow a **mod** 10
 Amíg auxB > 0 **és** aUtolsóSzej \neq auxB **mod** 10 **végezd el:**
 auxB \leftarrow auxB / 10
 vége(amíg)
 Ha auxB = 0 **akkor**
 // ha auxB 0 lett, az a aktuális számjegye nincs meg b-ben
 térítsd hamis
 vége(ha)
 auxB \leftarrow b *// visszaállítjuk b értékét*
 a \leftarrow a / 10 *// levágjuk az a szám utolsó számjegyet*
vége(amíg)
térítsd igaz
Vége(algoritmus)

Algoritmus azonos_3(x, y):
// vizsgáljuk, hogy x minden számjegye előfordul-e y-ban, és fordítva
térítsd AszámjegyeiBben(x, y) **és** AszámjegyeiBben(y, x)
Vége(algoritmus)

19. Legnagyobb palindrom

Adott az n ($0 < n < 2^{31}$) természetes szám. Határozzuk meg azt a **maxPal** számot, amely a legnagyobb palindrom, amelyet az n szám minden számjegyének átrendezése által kaphatunk. Ha nem lehet kialakítani a palindromot, amelyben az n szám minden számjegye szerepeljen, akkor **maxPal** értéke -1.

1. Példa: ha $n = 21523531$, akkor **maxPal** = 53211235.

2. Példa: ha $n = 12272351$, akkor **maxPal** = -1.

Legnagyobb palindrom

Elemzés

- Létrehozzuk az n számjegyeinek gyakoriságtömbjét, amelynek feldolgozása lehetővé teszi, hogy a kért számot hatékonyan generálhassuk
- Ha a számban csak páros gyakoriságok léteznek, elhelyezünk az új számba $\text{gyak}[i] / 2$ darab i számjegyet, értékeik csökkenő sorrendjében, majd elhelyezünk az új számba $\text{gyak}[i] / 2$ darab i számjegyet, értékeik növekvő sorrendjében
- Ha az i számjegy gyakorisága páratlan, és a szám közepének megfelelő hely még nem foglalt, ide teszünk egy i számjegyet
- Ha megjelenik még egy számjegy, amelynek gyakorisága páratlan, nem lehetséges palindromszámot generálni

Algoritmus palindrom(n):

Minden $i = 0, 9$ végezd el:

$gyak[i] \leftarrow 0$

vége(minden)

Amíg $n > 0$ **végezd el:**

$gyak[n \bmod 10] \leftarrow gyak[n \bmod 10] + 1$

$n \leftarrow n / 10$

vége(amíg)

$szám \leftarrow 0$ *// a legnagyobb palindromszám*

$közép \leftarrow -1$ *// a közép indexű hely még nincs elfoglalva*

$i \leftarrow 9$ *// a számjegyeket csökkenő sorrendben dolgozzuk fel*

...

Amíg $i \geq 0$ végezd el:

Ha $\text{gyak}[i] \bmod 2 = 1$ akkor

// az i számjegy páratlan számszor fordul elő

Ha $\text{közép} = -1$ akkor *// ha a közép indexű hely szabad*

$\text{közép} \leftarrow i$ *// elhelyezzük az i szj-t a szám közepére*

$\text{gyak}[i] \leftarrow \text{gyak}[i] - 1$ *// elhasználtunk egy i számjegyet*

$i \leftarrow i + 1$

különben

térítsd -1

// ha a közép indexű hely foglalt, nem

// tudunk palindromszámot generálni

vége(ha)

különben

$\text{hány} \leftarrow \text{gyak}[i] / 2$

// az i számjegyből hány darabot építünk a számba

...

Minden $j = 0$, hány - 1 végezd el:

$\text{szám} \leftarrow \text{szám} * 10 + i$

vége(minden)

$\text{gyak}[i] \leftarrow \text{gyak}[i] - \text{hány} \quad // \text{ hány darab } i\text{-t felhasználtunk}$

vége(ha)

$i \leftarrow i - 1 \quad // \text{ a következő (kisebb) számjegyérték}$

vége(amíg)

Ha közép $\neq -1$ akkor *// ha a közép kapott értéket*

$\text{szám} \leftarrow \text{szám} * 10 + \text{közép} \quad // \text{ beépítjük a számba}$

vége(ha)

Minden $i = 0, 9$ végezd el:

// most növekvő sorrendben dolgozzuk fel a számjegyeket

Minden $j = 0$, $\text{gyak}[i] - 1$ végezd el:

// minden számjegyet beépítünk annyiszor, ahány maradt

$\text{szám} \leftarrow \text{szám} * 10 + i$

térítsd szám

Vége(algoritmus)

2020.11.22.