Megoldott feladatok

Ionescu Klára

clara@cs.ubbcluj.ro

1. Leghosszabb prímszámokból álló tömbszakasz

*Írjunk algoritmust, amely egy természetes számokból álló számsorozatból meghatározza azt a **leghosszabb** tömbszakaszt, amely **csak prímszámokat** tartalmaz.*

Bemeneti adatok:

- ***n***: a számsorozat hossza
- ***x***: a tömb

Kimeneti adatok:

- ***start:*** a tömbszakasz első elemének indexe
- ***vége:*** a tömbszakasz utolsó elemének indexe

Funkció:

- Meghatározza azt a ***leghosszabb*** tömbszakaszt, amely ***csak prímszámokat*** tartalmaz.

Megoldás

- Szükségünk van egy függvényre, amely eldönti, hogy a paraméterként kapott érték prímszám-e vagy sem.
- A leghosszabb tömbszakasz meghatározását lineáris algoritmussal végezzük:
  - Átugrunk a sorozat elején található olyan számokon, amelyek nem prímszámok (ha léteznek)
  - A felfedezett első prímszám az első tömbszakasz első eleme; megjegyezzük a helyét
  - Haladunk, amíg az értékek prímszámok
  - Amikor megtaláljuk az első értéket, amely nem prímszám, aktualizáljuk a leghosszabb tömbszakasz hosszát
  - Folytatjuk a feldolgozást a sorozat végéig
- Kiírjuk a leghosszabb tömbszakasz elejének és végének indexeit.

2. Szigetek (BBTE, felvételi, 2016)

- Egy légitársaság az utasok rendelkezésére bocsátotta azoknak a földrajzi pontoknak a magasságait tartalmazó sorozatot, amelyek fölött a repülőgép száll majd Kolozsvár és New York között.
- Az ***a*** sorozatnak ***n*** darab ($3 \leq n \leq 10\,000$), 30 000-nél szigorúan kisebb természetes szám eleme van. A szárazföldnek megfelelő pontok magasságai 0-tól különböznek, míg az óceánnak megfelelő pontok magasságai egyenlők 0-val.
- *Sziget*nek olyan egymás utáni szárazföldnek megfelelő pontok sorozatát nevezzük, amely előtt és után víz található.
- A következő követelmények teljesítésére írjatok egy-egy algoritmust (alprogramot):

2. Szigetek (BBTE, felvételi, 2016)

- Határozzátok meg a szigetek számát.
- Határozzátok meg a leghosszabb sziget kezdetét, valamint a végét jelző pont sorszámát. Ha több megoldás létezik, csak egyet kell meghatároznotok. Ha nem létezik egyetlen sziget sem, akkor az eredmény 0 0.
- Döntsétek el, hogy a leghosszabb sziget *hegy* típusú-e vagy sem. Egy sziget *hegy* típusú, ha a felszínén található magasságok egy adott elemig szigorúan növekvő – nem üres – sorozatot alkotnak, majd szigorúan csökkenőt, amely nem üres.
- Döntsétek el, hogy a szárazföldnek megfelelő pontok magasságai páronként *különböző értékűek* vagy sem.
- Ha a 4. pontban feltett kérdésre a válasz „*nem*”, határozzátok meg a leggyakoribb magasság értékét és az előfordulásainak számát. Ha több ilyen magasság létezik, csak egyet kell megadnotok.

Specifikáció

Bemeneti adatok:

- ***n***: a magasságok darabszáma
- ***a***: a magasságok tömbje

Kimeneti adatok:

- ***szigetekSzáma***: szigetek száma
- ***eleje****:* a legnagyobb sziget kezdetének sorszáma
- ***vége****:* a legnagyobb sziget végének sorszáma
- ***magasság****:* a leggyakoribb magasság
- ***k****:* a leggyakoribb magasság előfordulásainak száma
- ***hegy****:* logikai változó, jelzi, hogy a sziget hegy-típusú vagy sem
- ***különbözők:*** logikai változó, jelzi, hogy a magasságok különbözők-e vagy sem

Funkciók: a feladat követelményeinek megfelelően

Alprogramok

beOlvas(n, a): beolvassa az ***n*** elemű a sorozatot

kiír(n, a): kiírja az ***n*** elemű ***a*** sorozatot

repülésSzimulálása(n, a, szigetekSzáma, eleje, vége): megkeresi a legnagyobb szigetet, amely az „***eleje***” pozíción kezdődik és a „***vége***” pozíción ér véget

hegyTulajdonság(eleje, vége, a): vizsgálja a "hegy" tulajdonságot

különbözők_e(n, a): vizsgálja a magasságok különbözőségét

leggyakoribbMagasság(n, a, magasság, k): meghatározza a leggyakoribb magasságot

kiírEredmény(szigetekSzáma, eleje, vége, hegy, különbözők, magasság, k)

3. Nyeregpontok

- Adott egy ***n*** soros és ***m*** oszlopos ($0 < n, m \leq 100$), egész számokat tároló kétdimenziós ***a*** tömb.
- Írjatok algoritmust, amely megszámolja a tömb nyeregpontjait!
- Egy a_{ij} elemet *nyeregpont*nak nevezünk, ha az a_{ij} elem legnagyobb a ***j.*** oszlopban és legkisebb az ***i.*** sorban, és fordítva.
- Az algoritmus bemeneti paraméterei ***n*** és ***a***, kimeneti paraméterek a nyeregpontok darabszáma (***k***) és az indexeik.
- ***Példa:*** Ha $n = 2$, $m = 6$ és $a = \begin{pmatrix} 5 & 2 & 8 & 4 & 9 & 3 \\ 7 & 1 & 6 & 3 & 8 & 5 \end{pmatrix}$ akkor $k = 2$, indexek: (1, 2) és (2, 5).

Specifikáció

Bemeneti adatok:

- ***n***, ***m***: a mátrix dimenziói
- ***a***: az adott mátrix

Kimeneti adatok:

- ***db***: nyeregpontok száma
- ***sorIndex***: a nyeregpontok sorindexeinek tömbje
- ***oszlopIndex***: a nyeregpontok oszlopindexeinek tömbje

Funkció:

- Meghatározza az ***a*** tömb nyeregpontjait

Alprogramok

A következő alprogramokat az ***a*** mátrix minden ***i.*** sorára meghívjuk.

- **minimumSoronként(a, n, m, i, sorokMinimumai, minSorHossza)**: meghatározzuk az **i.** sor minimának értékét és azokat az oszlopindexeket, ahol ez az érték előfordul (**sorokMinimumai**); ennek a sorozatnak hossza: **minSorHossza**
- **maximumOszloponkent(a, n, m, i, sorokMinimumai, minSorHossza, db, sorIndex, oszlopIndex, voltMár)**: bejárjuk azokat az oszlopokat (**sorokMinimumai**), ahol az **i.** sor egy-egy minimumértéke található, és meghatározzuk az oszlop maximumát; ha az oszlop maximuma az **i.** sorban található, akkor ez az elem nyeregpont; az indexeket tároljuk a **sorIndex** és **oszlopIndex** tömbökben; amikor az alprogramból kilépünk a tömb hossza **db**. A **voltMár** egy kétdimenziós tömb, amelyben megjegyezzük, hogy egy bizonyos elemet megtaláltunk már.

Alprogramok

- **maximumSoronkent(a, n, m, i, sorokMaximumai, maxSorHossza)**: meghatározzuk azokat az oszlopindexeket az **i.** sorban, ahol a sor maximumával egyenlő érték található (**sorokMaximumai**); ennek a sorozatnak hossza: **maxSorHossza**
- **minimumOszloponkent(a, n, m, i, sorokMaximumai, maxSorHossza, db, sorIndex, oszlopIndex, voltMár)**: bejárjuk azokat az oszlopokat (**sorokMaximumai**), ahol az **i.**. sor egy-egy "maximumértéke" található, és meghatározzuk az oszlop minimumát; ha az oszlop minimuma az **i.** sorban található, akkor ez nyeregpont; az indexeket tároljuk a **sorIndex** és **oszlopIndex** tömbökben; amikor az alprogramból kilépünk a tömb hossza **db** (folytattuk a tárolást).
- **nyeregpontok(a, n, m, db, sorIndex, oszlopIndex)**: ez az alprogram hívja az előbbieket; amikor kilépünk **db** értéke = nyeregpontok száma, a **db** elemű **sorIndex** és **oszlopIndex** tömbök a nyeregpontok indexeit tartalmazzák.

4. Maximális szorzat (felvételi, 2016)

- Adott az ***n*** elemű ($3 \leq n \leq 10\,000$), egész számokat tároló ***x*** sorozat, ahol az elemek értéke nagyobb, mint -30 000 és kisebb, mint 30 000.
- Írjatok alprogramot, amely meghatároz *három* számot az ***x*** sorozatból, amelyeknek a szorzata *maximális*.
- Az alprogram bemeneti paraméterei ***n*** és ***x***, kimeneti paraméterei az ***a***, ***b*** és ***c*** számok, amelyek az ***x*** sorozat elemei és rendelkeznek a kért tulajdonsággal. Ha a feladatnak több megoldása van, csak egyet kell megadnotok.
- ***Példa:*** ha ***n*** = 10 és ***a*** = (3, -5, 0, 5, 2, -1, 0, 1, 6, 8), a három szám:

a = 5, ***b*** = 6, ***c*** = 8.

Specifikáció

Bemeneti adatok:

- ***n**:* a tömb hossza
- ***x**:* a tömb

Kimeneti adatok:

- ***a***, ***b***, ***c***: három szám a tömb elemei közül, amelyeknek a szorzata maximális

Funkció:

- Meghatározza azt a *három* számot az ***x*** sorozatból, amelyeknek a szorzata *maximális*.

Elemzés

- Több megközelítés is lehetséges, természetesen, különböző bonyolultságú algoritmussal

1. **Naiv algoritmus:** generálunk minden lehetséges indexhármast, kiszámítjuk a megfelelő elemek szorzatát és megőrizzük a szorzatok közül a legnagyobbat
   - Ha az értékek a megengedett felső határ közelében találhatók, a három szám szorzata túlcsordulhat
   - Bonyolultság: $O(n^3)$
2. **Rendezéssel:** a sorozat rendezhető **lineáris rendezés**sel, majd feldolgozzuk a három legnagyobb és két legkisebb értéket
   - Nehézséget okoz a tény, hogy a számok értéke nagyobb, mint -30 000 és kisebb, mint 30 000
   - Bonyolultság: $O(n)$
3. **Rendezéssel:** a sorozat rendezhető **buborékrendezés**sel
   - Bonyolultság: $O(n^2)$

Elemzés

4. Lineáris algoritmussal (Ez a jó megoldás!!!):

- Meghatározzuk a tömb maximumát (**max1**)
- Meghatározzuk a tömb második maximumát (**max2**)
- Meghatározzuk a tömb harmadik maximumát (**max3**)
- Meghatározzuk a tömb minimumát (**min1**)
- Meghatározzuk a tömb második minimumát (**min2**)
- Feldolgozzuk ezt az öt értéket
- Bonyolultság: $5 * O(n) + O(1) = O(n)$

Az öt érték feldolgozása

```
…
Ha max1 > 0 és min1 * min2 > max2 * max3 akkor
        a ← max1
        b ← min1
        c ← min2
különben
        a ← max1
        b ← max2
        c ← max3
vége(ha)
…
```

5. Balra át

- Szeptember 15-én az iskola igazgatója felkéri az első osztályos gyermekeket, hogy forduljanak arccal feléje, majd abból a célból, hogy felvezethesse őket az osztálytermekbe, kiadja a parancsot: „Balra át!” A gyermekek bizonytalanok. Van, aki balra fordul, van, aki jobbra. Aki szemben találja magát a mellette állóval, azt hiszi, hogy rosszul fordult és egy időegység alatt egyszer sarkon fordul. Adott egy bizonyos kezdeti konfiguráció, amely a „Balra át!” parancs után alakult ki. Állapítsátok meg, **hány időegység** alatt „nyugszik meg” a sor. (Legtöbb 255 gyerekkel dolgozhatunk.)
- *Példa:* ha ***n* = 5**, a parancs utáni konfiguráció: **bjbjb**, ahol **b** balra fordult gyereket, **j** jobbra fordult gyereket jelent, az eredmény: **2**

Elemzés

- Amikor két gyermek egymással szemben találja magát, a két gyermeket egy **jb** karakterpáros ábrázolja.
- Az algoritmusban ezt a karakterpárost felülírjuk **bj**-vel.
- Ekkor előfordulhat, hogy a **j**-vel ábrázolt gyermek mellett egy **b**-vel ábrázolt gyermek áll, de most már nem szabad felülírni az értékeiket, mivel a szövegben ki van kötve, hogy egy időegység alatt egy gyermek csak egyszer fordul meg.
- Ezt két féleképpen is megoldhatjuk:
  - Felhasználunk egy „másolatot” a sorozatról, amelyen végigmegyünk, keressük a **jb** párokat de a módosításokat az eredeti sorozaton végezzük; végül a másolatra rámásoljuk az eredetit.
  - A másolat használata elkerülhető, ha abban az esetben, amikor két gyermek állapota megváltozik, a ciklusváltozót 2-vel növeljük

A legfontosabb észrevétel

- Az algoritmus hasonlít a buborékrendezéshez!
- Különbségek:
  - Nem szükséges felcserélni a két elemet, felülírjuk ezeket.
  - Vagy készítünk ideiglenes másolatot a gyermekek sorozatáról, vagy minden felülírás után kettőt lépünk előre; ebben az esetben Amíg struktúrával dolgozunk a Minden helyett.
- Az időt csak akkor növeljük, ha volt fordulás, mivel az utolsó bejárást csak ellenőrzés céljából végezzük.

```
Algoritmus balraÁt(n, gy):                    // Négyzetes bonyolultság
        időEgység ← 0
        Ismételd
                i ← 1
                fordult ← hamis
                Amíg i ≤ n végezd el:
                        Ha gy_i = 'j' és gy_(i + 1) = 'b' akkor
                                gy_i ← 'b'
                                gy_(i + 1) ← 'j'
                                i ← i + 2
                                fordult = igaz
                        különben
                                i ← i + 1
                        vége(ha)
                vége(amíg)
                Ha fordult akkor időEgység ← időEgység + 1
                vége(ha)
        ameddig nem fordult
        térítsd időEgység
Vége(algoritmus)
```

```
Algoritmus balraÁt (n, gy):                                   // lineáris algoritmus
        időEgység ← 0; idő ← 0; lépés ← 0
        Minden i = 1, n végezd el:
                Ha gy_i = 'j' akkor
                                idő ← idő + 1
                                Ha lépés > 0 akkor lépés ← lépés – 1 vége(ha)
                vége(ha)
                Ha gy_i = 'b' akkor
                        Ha idő > 0 akkor
                                        időEgység ← idő + lépés
                                        lépés ← lépés + 1
                        vége(ha)
                vége(ha)
        vége(minden)
        térítsd időEgység
Vége(algoritmus)
```

6. Maximális összegű leghosszabb tömbszakasz

Adott egy ***n*** egész számból álló számsorozat, amely biztosan tartalmaz legalább egy pozitív számot. Írjunk programot, amely meghatározza azt a *leghosszabb tömbszakaszt*, amelynek *összege a lehető legnagyobb*.

Példa: ***n* = 10**, a sorozat: **1, 2, –6, 3, 4, 5, –2, 10, –5, –6**. Megoldjuk papíron:

- Maximális összeg: **20**.
- A tömbszakasz hossza: **5**
- A tömbszakasz első elemének sorszáma: **4**
- A tömbszakasz utolsó elemének sorszáma: **8**
- A keresett tömbszakasz: **3, 4, 5, –2, 10**.

6. Maximális összegű leghosszabb tömbszakasz

1. Megoldás

- Generálunk minden *bal* és *jobb* egész számpárt, amelyekre:
- $1 \leq bal \leq jobb \leq n$, és kiszámítjuk a megfelelő $t_{bal}, t_{bal+1}, \ldots, t_{jobb}$ tömbszakaszok összegét. Közben, kiválasztjuk azt a tömbszakaszt, amelynek összege maximális.
- Ennek az algoritmusnak a bonyolultsága $O(n^3)$ mivel három egymásba ágyazott ciklusból áll.
- Ez a bonyolultság elfogadhatatlan, ezért előbb keresünk egy négyzetes algoritmust, majd egy lineárist!

```
Algoritmus MaxÖsszegűTömbszakasz_1(n, t, MaxS):
                                        { bemeneti adatok: n, t; kimeneti adat: MaxS }
    MaxS ← t1                                   { a keresett maximális összeg }
    Minden bal = 1, n végezd el:          { bal: a tömbszakasz első elemének indexe }
        Minden jobb = bal, n végezd el:{ jobb: a tömbszakasz utolsó elemének indexe }
            össz ← 0                      { össz: az aktuális tömbszakasz összege }
            Minden i = bal, jobb végezd el:
                össz ← össz + ti
            vége(minden)
            Ha MaxS < össz akkor
                MaxS ← össz { stb... }
            vége(ha)
        vége(minden)
    vége(minden)
Vége(algoritmus)
```

6. Maximális összegű leghosszabb tömbszakasz

2. Megoldás

- Észrevesszük, hogy a $t_{bal}, t_{bal+1}, \ldots, t_{jobb}$ tömbszakasz összegét ki lehet számítani az előző lépésben kiszámolt $t_{bal}, t_{bal+1}, \ldots, t_{jobb-1}$ tömbszakasz összegének segítségével.
- Minden $t_{bal}, t_{bal+1}, \ldots, t_{jobb}$-nak megfelelő összeget összehasonlítjuk az eddig megtalált maximális összeggel azonnal a kiszámolás után.
- Az algoritmusunk így négyzetes lesz, mivel csak két egymásba ágyazott ciklust fog tartalmazni:

```
Algoritmus MaxÖsszegűTömbszakasz_2(n, t, MaxS):
        MaxS ← t1       { bemeneti adatok: n, t; kimeneti adat: MaxS }
        Minden bal=1,n végezd el:
                össz ← 0
                Minden jobb = bal, n végezd el:
                        össz ← össz + tjobb
                        Ha MaxS < össz akkor
                                MaxS ← össz… stb.
                        vége(ha)
                vége(minden)
        vége(minden)
Vége(algoritmus)
```

6. Maximális összegű leghosszabb tömbszakasz

4. Megoldás

- ***A feladat garantálja, hogy a sorozatban van legalább egy pozitív szám!***
- Következik, hogy, ha a sorozatnak $n - 1$ negatív eleme és egyetlen pozitív eleme lenne, akkor a maximális összegű tömbszakasz ebből az egyetlen pozitív számból állna.
- Következik, hogy amíg egy aktuális összeg pozitív, addig az ***AktMax***-hoz hozzáadjuk a t_i-t.
- Ha, egy adott pillanatban ***AktMax*** negatívvá válik, akkor ***AktMax*** új értéke t_i lesz.

```
Algoritmus MaxÖsszegűTömbszakasz_3(n, t, eleje, vége, MaxS):
                                    { bemeneti adatok: n, t; kimeneti adatok: MaxS, eleje, vége }
    MaxS ← t1; összeg ← MaxS; kezd ← 1; eleje ← 1; vége ← 1
    Minden i = 2, n végezd el:
        Ha összeg < 0 akkor
            összeg ← ti; kezd ← i
        különben összeg ← összeg + ti
        vége(ha)
        Ha MaxS < összeg akkor
            MaxS ← összeg; eleje ← kezd; vége ← i
        különben
            Ha MaxS = összeg és vége – eleje < i – kezd akkor
                eleje ← kezd; vége ← i
            vége(ha)
        vége(ha)
    vége(minden)
Vége(algoritmus)
```