Minimum/maximum kiválasztás

Ionescu Klára

clara@cs.ubbcluj.ro

Megkeresitek a honlapon:

http://www.cs.ubbcluj.ro/felkeszitok-programja-es-tematikaja-a-matematika-es-informatika-karra-felvetelizo-kozepiskolas-diakok-szamara-2019/

Bevezetés

- A feladatok ***feladatosztályok***ba sorolhatók a jellegük szerint.
- E feladatosztályokhoz készítünk a teljes feladatosztályt megoldó ***algoritmusosztály***t.
- Ezeket az algoritmusosztályokat ***programozási tétel***eknek nevezzük (lásd 1. konzultáció).
- Bebizonyítható, hogy a megoldások a feladat ***garantáltan helyes és optimális*** megoldásai.
- A bemenet és a kimenet szerint négy csoportra oszthatók:
  - sorozathoz érték rendelése (**1 sorozat – 1 érték**)
  - sorozathoz sorozat rendelése (**1 sorozat – 1 sorozat**)
  - sorozatokhoz sorozat rendelése (**több sorozat – 1 sorozat**)
  - sorozathoz sorozatok rendelése (**1 sorozat – több sorozat**)

Feladatok

1. Egy kórházban megmérik minden reggel a betegek hőmérsékletét. Határozzuk meg a ***leglázasabb*** beteg hőmérsékletét!
2. Egy táblázatban feltüntették az elsőéves informatika szakos egyetemisták nevét a felvételi után, a vizsgaátlaguk sorrendjében. Határozzuk meg annak az egyetemistának a nevét, akivel ***az ábécé sorrendű táblázat kezdődni fog***!
3. A szociális ösztöndíj kiosztása céljából összegyűjtötték az egyetemisták családjainak jövedelmére vonatkozó adatokat. Határozzuk meg a ***legkisebb*** jövedelmet!

Elemzés

- A megoldások hasonlóak lesznek, mivel mindegyikben meg kell határoznunk az adatok között található ***legnagyobb***, illetve ***legkisebb*** értéket.
- Előfordulhat, hogy több ilyen legnagyobb, illetve legkisebb elem létezik.
- A **Kiválasztás(n, X, sorszám)** programozási tétel sajátos esetével állunk szemben, amikor a keresett elem az adott sorozat legkisebb/legnagyobb értékű eleme.
- Az algoritmus mindig **Minden** típusú struktúrával dolgozik, hiszen minden adatot meg kell vizsgálnunk.

Maximumkiválasztás programozási tétel

- Adott az ***N*** elemű ***X*** sorozat. Határozzuk meg a sorozat ***legnagyobb*** (vagy legkisebb) értékét (vagy sorszámát)!

Elemzés

- Előfordulhat, hogy ***több*** legnagyobb elem létezik, de nekünk most az értéket (vagy az első előfordulás helyét) kell meghatároznunk.
- A megoldásban ***minden*** adatot meg kell vizsgálnunk, ezért az algoritmus, egy **Minden** típusú struktúrával dolgozik.
- A max segédváltozó a sorozat első elemétől kap kezdőértéket.

Algoritmus Maximumkiválasztás(N, X, max):

{ *Bemeneti adatok: az* N *elemű* X *sorozat* }

{ *Kimeneti adat:* max, *a legnagyobb elem értéke* }

{ *Funkció: meghatározza az* N *elemű* X *sorozat legnagyobb értékét* }

max ← X_1

Minden i = 2, n **végezd el**:

Ha max < X_i **akkor**

max ← X_i

vége(ha)

vége(minden)

Vége(algoritmus)

1_MaximumKivalasztas.cpp

```
Algortmus maximumKiválasztásRek(n, X):
                                                    { rekurzív implementáció }
                                    { Bemeneti adatok: az N elemű X sorozat }
                                    Kimeneti adat: max, a legnagyobb elem értéke }
                    { Funkció: meghatározza az N elemű X sorozat legnagyobb értékét }
   Ha n = 1 akkor
        térítsd X1
   különben
        Ha Xn > maximumKiválasztásRek(n - 1, X) akkor
              térítsd Xn
        különben
              térítsd maximumKiválasztásRek(n - 1, X)
                                    { vizsgálni fogjuk a sorozat következő elemét }
        vége(ha)
   vége(ha)
Vége(algoritmus)
```

Az algoritmus hívása: **Ki:** maximumKiválasztásRek(n, X)

Észrevételek

- A maximumot/minimumot tartalmazó segédváltozónak ***az adatok közül választunk kezdőértéket***, mivel így nem áll fenn a veszély, hogy az algoritmus eredménye egy, az adataink között nem létező érték legyen.

- Ha az adatokat, amelyekből a maximumkiválasztást végezzük valamilyen számolás eredményeként kapjuk, akkor (ritkán) kereshetünk egy olyan értéket, amelynél biztos lesz nagyobb a kiszámolt értékek között.

Változat a maximumkiválasztásra

- Az előbbi algoritmus a maximum ***értékét*** határozza meg.
- Ha nem az érték, hanem a ***hely*** érdekel (index), ahol ***elsőként*** fedeztük fel a legnagyobb értéket, akkor az algoritmus a következőképpen alakul:

Algoritmus Maximum_helye(N, X, hely):
{ *Bemeneti adatok: az* N *elemű* X *sorozat* }
{ *Kimeneti adat:* hely, *a legnagyobb elem pozíciója* }
{ *Funkció: meghatározza a maximumérték első előfordulásnak helyét* }
hely ← 1 { hely *az első elem pozíciója* }
Minden i = 2, n **végezd el:**
Ha $X_{hely} < X_i$ **akkor**
hely ← i { *a maximális elem helye (pozíciója)* }
vége(ha)
vége(minden)
Vége(algoritmus)

2_MaximumHelye.cpp

```
Algoritmus maximumHelyeRek(n, X):
                                        { Bemeneti adatok: az N elemű X sorozat }
                                        { Kimeneti adat: hely, a legnagyobb elem pozíciója }
                                        { Funkció: meghatározza a maximumérték első előfordulásnak helyét }
                                        { rekurzív implementáció }
    Ha n = 1 akkor
        térítsd 1
    különben
        Ha Xn > XmaximumHelyeRek(n - 1, X) akkor
            térítsd n
        különben
            térítsd maximumHelyeRek(n - 1, X)
        vége(ha)
    vége(ha)
Vége(algoritmus)
```

Az algoritmus hívása: **Ki:** maximumHelyeRek(n, X)

Észrevétel

- Ha ***nem az első***, maximumot tároló helyet kell meghatároznunk, hanem az ***utolsót***, bejárhatjuk a ***sorozatot fordított*** irányban (az előző algoritmust használva).
- Alkalmazhatjuk az előbbi algoritmust úgy is, hogy a **„<”** relációs műveletet **„≤”**-re cseréljük.

Minden maximum helye

Határozzuk meg az ***összes*** olyan indexet, amely indexű elemek egyenlők a legnagyobb elemmel!

Első ötlet:

1. ***Megállapítjuk*** ***a maximum értékét***.
2. *Bejárjuk* ***újra*** *a sorozatot*, abból a célból, hogy kiírhassunk minden indexet, amelyhez ez a legnagyobb érték tartozik.

Algoritmus Minden_max(N, X, db, indexek):

{ *Bemeneti adatok: az* N *elemű* X *sorozat* }

{ *Kimeneti adat: a* db *elemű* indexek *sorozat* }

{ *Funkció: meghatározza minden legnagyobb elem helyét* }

max ← X_1

Minden i = 2, N **végezd el:**

Ha max < X_i **akkor** max ← X_i

vége(ha)

vége(minden)

db ← 0

Minden i = 1, N **végezd el:**

Ha max = X_i **akkor**

db ← db + 1

$indexek_{db}$ ← i

vége(ha)

vége(minden)

Vége(algoritmus)

2. ötlet

Ha ez a megoldás nem megfelelő, mivel

- az adott tömböt kétszer kell bejárnunk, vagy
- olyan feladatunk van, ahol ***a maximumhoz tartozó adatok egy másik (esetleg bonyolult) algoritmus végrehajtásának eredményei***,

$\Rightarrow$ írhatunk olyan algoritmust, amely ***csak egyszer járja be a sorozatot***:

Algoritmus Minden_max_2(N, X, db, indexek):

{ *Bemeneti adatok: az* N *elemű* X *sorozat* }

{ *Kimeneti adat: a* db *elemű* indexek *sorozat* }

{ *Funkció: meghatározza minden legnagyobb elem helyét* }

max ← X_1 db ← 1, $indexek_1$ ← 1

Minden i = 2, N **végezd el:**

Ha max < X_i **akkor**

max ← X_i

db ← 1

$indexek_{db}$ ← i

különben

Ha max = X_i **akkor**

db ← db + 1

$indexek_{db}$ ← i

vége(ha)

vége(ha)

vége(minden)

Vége(algoritmus) **3_MindenMaximumHelye.cpp**

```
Algoritmus maximumokHelyeRek(n, X, poz, db, max):
        Ha n ≥ 1 akkor
                Ha max < X_n akkor
                        poz_1 ← n
                        db ← 1
                        maximumokHelyeRek(n - 1, X, poz, db, X_n)
                különben
                        Ha max = X_n akkor
                                db ← db + 1
                                poz_db ← n
                                maximumokHelyeRek(n - 1, X, poz, db, max)
                        különben
                                maximumokHelyeRek(n - 1, X, poz, db, max)
                        vége(ha)
        vége(ha)
Vége(algoritmus)
```

Az algoritmus hívása: maximumokHelyeRek(n, X, poz, db, X_n), ahol előbb db ← 0

Megoldott feladatok
1. Leghosszabb 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.

..\Feladatok\1_LeghosszabbTömbszakaszPrimek\maxTombszakaszPr.cpp

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)

..\Feladatok\2_Szigetek\Szigetek.cpp

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.

..\Feladatok\3_Nyeregpontok\nyeregpontok.cpp

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)
…

..\Feladatok\4_Max_Szorzat\maxSzorzat.cpp