Az algoritmus leírása, helyesség, lépésszám

Csima Judit
BME SZIT
csima@cs.bme.hu

2019. szeptember 11.

Mit értünk algoritmus alatt?

Az algoritmus fogalmára létezik pontos, formális definíció, de ebben a tárgyban beérjük a következő, a gyakorlatban jól használható meghatározással:

1. adott formátumú inputból adott formátumú elvárt kimenetet előállító
2. mechanikusan végrehajtható utasítássorozat, ami
3. minden megfelelő formátumú inputon véges sok lépésben végetér.

Példák lehetséges helyzetekre, amikor algoritmusra van szükségünk:

1. példa Az input egy $n > 0$ darab pozitív egész számot tartalmazó A tömb, az elvárt kimenet a legkisebb, a tömbben szereplő szám

2. példa Az input egy $n > 0$ darab pozitív egész számot tartalmazó A tömb, az elvárt kimenet 0, ha a tömbben nincs 7-es szám és 1, ha van

Kitérő Egy n méretű tömb alatt azt értjük, hogy van n darab cella, megszámozva 0-tól $n-1$-ig, ahol minden cellában egy érték szerepel. Az értékeket az index alapján egy lépésben ki tudjuk olvasni. Az n hosszú A tömbre az $A[0 : n-1]$ jelölést fogjuk használni, az i indexű cella értékére pedig $A[i]$ jelöléssel hivatkozunk.

Amikor egy algoritmust megadunk, akkor három dolgot kell megtennünk:

1. leírni az algoritmust
   - magyarul, szövegesen, precízen VAGY
   - pszeudokóddal
2. megindokolni, hogy az algoritmus helyes, a kívánt kimenetet adja
3. megbecsülni, hogy hány lépést tesz az algoritmus (erről majd a jövő órán lesz szó részletesebben):
   - a lépésszám függ az input méretétől
   - minden fajta feladatnál megmondjuk előre, hogy mely lépések a lényegesek, miket kell megszámolni
   - csak felső becslést tudunk mondani a legtöbb esetben

Algoritmusok arra, hogy van-e 7-es szám egy tömbben

A korábban, a 2. példában látott feladatot oldjuk meg több algoritmussal, ezeken gyakoroljuk az algoritmus leírását, a helyesség belátását és a lépésszám becslését.

1. algoritmus szövegesen
Végigmegyünk a tömbbön az elejétől a végéig és minden cellában megnézzük, hogy 7-es van-e ott. Ha látunk valahol 7-est, akkor megállunk és azt mondjuk, hogy "Van!", ha végigérünk a tömbön anélkül, hogy látnánk 7-est, akkor azt mondjuk a végén, hogy "Nincs!". (Az eredeti példában 0 vagy 1 kimenetet vártunk, ezért ez az algoritmus nem pont abban a formában oldja meg a feladatot, de a lényeg ugyanaz.)

1. algoritmus pszeudokóddal

```
ciklus i = 0-tól n-1-ig:
    ha A[i] == 7:
     return ‘‘Van!’’ és stop
ciklus vége
return ‘‘Nincs!’’
```

A pszeudokódban a következő jelöléseket használjuk:

- == jelöli azt, hogy két értékről megnézzük, hogy egyenlők-e, összehasonlítva őket
- a ciklus elejét és végét jelöljük, a köztük levő rész a ciklus magja, ami (ebben az esetben) $i = 0, 1, 2, \ldots, n-1$ értékekre fut le

Az 1. algoritmus helyes mert minden cellát megnéztünk és akkor és csak akkor adunk "Van!" kimenetet, ha találunk 7-es értéket
Az 1. algoritmus lépésszáma: legfeljebb n összehasonlítás történik és további egy lépés a végén az eredmény közlése

2. algoritmus pszeudokóddal

```
van_e_hetes := 0
ciklus i = 0-tól n-1-ig:
    ha A[i] == 7:
       van_e_hetes := 1
ciklus vége
return van_e_hetes
```

A pszeudokódban a következő új jelöléseket használjuk:

- itt a van_e_hetes egy változó, ami 0 vagy 1 értéket tárol
- := jelöli azt, hogy egy változónak értéket adunk

A 2. algoritmus helyes mert ismét csak minden cellát megnéztünk és pontosan akkor állítjuk át a kimenetet 1-re, ha találunk 7-es értéket
A 2. algoritmus lépésszáma: egy értékadás az elején, legfeljebb n összehasonlítás és legfeljebb n értékadás a ciklus futása során és további egy lépés a végén az eredmény közlése

Megjegyzés: A következő órán részletesen megbeszéljük, hogy hogyan szokás mérni a lépésszámot, a lényeg az lesz, hogy csak bizonyos lépéseket fogunk számolni, nem az összeset. A mostanihoz hasonló helyzetekben például csak az összehasonlítások fognak számítani majd (meg a mozgatások, de ezek itt nem voltak, ezekről majd később beszélünk).

Algoritmusok pozitív egészeket tartalmazó tömb legkisebb elemének megkeresésére

A korábban, az 1. példában látott feladatot oldjuk meg most több algoritmussal.

3. (nagyon-nagyon béna) algoritmus szövegesen
Végigmegyünk a tömbbön az elejétől a végéig és megnézzük, hogy van-e bárhol 1-es (ezt az 1. vagy 2. algoritmushoz hasonlóan tesszük). Ha van, akkor a kimenet 1 és leállunk. Ha végigérünk a tömbön anélkül, hogy látnánk 1-est, akkor megcsináljuk ugynezt a 2-es értékkel. Ezt addig folytatjuk, amíg olyan értékhez érünk, ami végre valahára szerepel a tömbben.

3. (nagyon-nagyon béna) algoritmus pszeudokóddal

```
j := 1
ciklus
    ciklus i = 0-tól n-1-ig:
         ha A[i] == j:
             return j és stop
    ciklus vége
    j := j +1
ciklus vége
```

Itt két egymásba ágyazott ciklust használunk, a külső ciklus annyiszor fog lefutni, amekkora a legkisebb szereplő szám értéke.

Ez az algoritmus több szempontból is nagyon béna:

- A lépésszám nem csak a tömb méretétől függ, hanem a tömbben szereplő számok nagyságától is, ezt nem szeretjük.
- A pszeudokódos változatban a külső ciklusnál nincs feltétel arra, hogy meddig fut, se fix érték, hogy hányszor fut le legfeljebb. Ilyen konstrukcióval könnyű soha le nem álló kódot írni (bár itt nem ez a helyzet).

A 3. algoritmus helyes amúgy, mert pontosan akkor fog leállni, amikor a legkisebb tömbbeli elem értékét eléri a j változó.
A 3. algoritmus lépésszáma: legfeljebb n-szer a tömbben szereplő legkisebb értéknyi összehasonlítást használunk (a következő órán megbeszéljük, hogy ebben a helyzetben csak az összehasonlításokat akarjuk megszámolni), mert a belső ciklus minden lefutása legfeljebb n összehasonlítást jelent és a belső ciklus legfeljebb annyiszor fut le, amekkora a legkisebb érték a tömbben.

Feladat Módosítsuk a fenti pszeudokódot úgy, hogy ne csak a legkisebb értéket írja ki a végén, hanem azt a tömbbeli indexet is, ahol ez megtalálható. (Ha több ilyen van, akkor elég egy ilyen indexet megadni.)

4. (még mindig nagyon-nagyon béna) algoritmus pszeudokóddal

```
jelölt := 0
min_hely: = -1
megvan: = 0

ciklus amíg megvan == 0:
    jelölt : = jelölt + 1
    ciklus i = 0-tól n-1-ig:
         ha A[i] == jelölt:
             megvan := 1
             min_hely: = i
    ciklus vége
ciklus vége
return jelölt és return min_hely
```

Itt megint két egymásba ágyazott ciklust használunk, a külső ciklus minden lefutása előtt ellenőrizzük a feltételt (megvan == 0 fennáll-e) és csak akkor indul el a ciklus következő lefutása, ha ez igaz.

Ez az algoritmus, bár kissé máshogy működik, mint az előző, de továbbra is béna (noha helyes, ugyanazért, amiért az előző is helyes volt), mert a lépésszáma ugyanolyan rossz, mint annak.

Feladat Ha egy tömbben több helyen is szerepel a legkisebb érték, akkor melyiknek az indexét fogja kiírni az algoritmus? Futassa le az algot (fejben vagy papíron) a 3,2,10,2,10 tömbre, ez segít a kérdés megválaszolásában.
Mi lenne, ha a külső ciklus első sorát (jelölt := jelölt + 1) a belső ciklushoz tartozó ciklus vége sor mögé mozgatnánk? Helyes maradna akkor is az algoritmus?

5. algoritmus (?) a legkisebb elem megkeresésére, szövegesen

Rendezzük a tömböt, ezután a legkisebb elem a legelső cellában lesz.

Ezzel az a baj, hogy ez így, ebben a formában nem algoritmus, amíg nem mondjuk meg, hogy hogyan kell egy tömb elemeit növekvően rendezni. Ahhoz, hogy ezt algoritmusnak lehessen nevezni ezt is meg kéne mondani pontosan (és csak ezután lehetne lépésszámról beszélni).

6. algoritmus a legkisebb elem megkeresésére

```
min  := A[0]
ciklus i = 1-tól n-1-ig:
    ha A[i] < min:
        min := A[i]
ciklus vége
return min
```

A 6. algoritmus helyes mert a min változóba bekerül a tömb legkisebb értéke akkor, amikor elérünk hozzá, később pedig soha nem írjuk már át ezt a változót.

A 6. algoritmus lépésszáma: egy értékadás az elején, $n-1$ összehasonlítás, legfeljebb $n-1$ értékadás és további egy lépés a végén az eredmény közlése

Feladat: Hogyan kéne ezt a kódot módosítani, hogy a legkisebb elem értékén kívül annak az indexét is kiírja?

7. algoritmus a legkisebb elem megkeresésére

```
ciklus i = 0-tól n-2-ig:
    ha A[i] < A[i+1]:
        cseréljük meg A[i]-t és A[i+1]-et
ciklus vége
return A[n-1]
```

A 7. algoritmus helyes mert a legkisebb elemet onnantól kezdve, hogy elérjük végig jobbra mozgatjuk, így a végén a tömb utolsó cellájába kerül.
A 7. algoritmus lépésszáma: $n-1$ összehasonlítás, legfeljebb $n-1$ csere és további egy lépés a végén az eredmény közlése

Feladat, nem nagyon könnyű: Hogyan kéne ezt a kódot módosítani, hogy a legkisebb elem értékén kívül annak az indexét is kiírja?