Elemi algoritmusok – prefix összegzés

Menyhárt László[1] Gábor, Zsakó László[2]
(ORCID: 0000-0002-1574-4454, 0000-0002-4614-1509)

[1]menyhart@inf.elte.hu, [2]zsako@caesar.elte.hu
ELTE Eötvös Loránd Tudományegyetem, Budapest
Informatikai Kar

Absztrakt. Ebben a cikkünkben egy speciális problémát, a prefix összegzést (más néven kumulatív összegzést), és annak variációit mutatjuk be. Részletesen megvizsgáljuk az alapproblémát és összehasonlítjuk a lehetséges megoldásaikat. Hallgatóink hasonló problémákat oldottak meg versenyeken, így a megoldásaikról statisztikával rendelkezünk. A cikket olyan olvasóknak ajánljuk, akik érdeklődnek a programozási módszertan iránt vagy diákjaik versenyeken fognak indulni.

Kulcsszavak: módszeres programozás, oktatás, mérés, prefix összeg, kumulatív összeg.

1. Bevezetés

A programozás tanítás során az algoritmikus struktúrák (szekvencia, elágazás, ciklus) bevezetése után általában alapvető algoritmusokkal (a magyar szakirodalomban programozási tételekkel, típusalgoritmusokkal) foglalkoznak, mint pl. összegzés, keresés, maximumkiválasztás, rendezés stb.

A programozási versenyeken (és a tehetséggondozásban) megjelennek feladatmegoldási stratégiák (mint. pl. mohó stratégia, dinamikus programozás), a kettő közötti űrt azonban kevés helyen töltik ki. Ezt az űrt töltötte ki J. Bentley klasszikus cikksorozata és könyve (Programming Pearls [1], magyarul *A programozás gyöngyszemei* címen jelent meg) 1984-ből, újabban pedig például az algoritmizálást tanító weboldalak [2,3]. Az első megjelenése azonban 1954-es, egy nem közvetlen informatika témájú folyóiratban [4].

Ebben a cikkben azt járjuk körbe, hogy mihez kezdenek a kumulatív összegzés (prefix összegzés) feladattípussal magyar programozási versenyek résztvevői. Érdekességként két nagyon különböző korosztály vizsgálunk, az egyikben 13-16 éves diákok szerepelhetnek – Nemes Tihamér Online Programozási Verseny [5], a másikban pedig egyetemen informatikus szakra járó, a programozással most ismerkedő egyetemi hallgatók – Tehetségkutató Egyetemi Programozási Verseny [6].

Nem szándékozunk ebben a cikkben az algoritmus haladó alkalmazásaival foglalkozni (kiegyensúlyozott bináris fákkal, Fenwick fákkal, ...).

2. Kumulatív összegzés alapfeladat

Feladat Adott egy `N` elemű pozitív és negatív számokat is tartalmazó számsorozat. Adjuk meg a sorozat azon `[a,b]` intervallumát, ahol az elemek összege maximális! Olyan `a` és `b` indexet kell megadnunk, amire az alábbi feltétel teljesül:

$$1 \le a \le b \le N \quad \text{és} \quad \forall p,q\left(1 \le p \le q \le N\right): \quad \sum_{i=a}^{b} X_i \ge \sum_{i=p}^{q} X_i$$

A megoldás

Az alapmegoldás kiszámolja minden intervallumra a számok összegét, majd ezekre maximumkiválasztást végez:

```
összeg(i,j):
  S:=0
  Ciklus k=i-től j-ig
    S:=S+X[k]
  Ciklus vége
  összeg:=S
Függvény vége.
```

```
Alapmegoldás(max,a,b):
  max:=-∞
  Ciklus i=1-től N-ig
    Ciklus j=i-től N-ig
      s:=összeg(i,j)
      Ha s>max akkor max:=s; a:=i; b:=j
    Ciklus vége
  Ciklus vége
Eljárás vége.
```

A három ciklus miatt a futási ideje N^3-bel arányos. Gyakran alkalmazható ötlet, hogy először nem azt számoljuk ki, amit a feladat kér, hanem valami egyszerűbbet, amiből viszont már könnyű megkapni a feladat megoldását.

Számítsuk ki az `s` vektorba a sorozat első `i` elemének összegét!

$$\forall i(0 \le i \le N): \quad s(i) = \sum_{j=1}^{i} x_j$$

ami rekurzív összefüggést használva, egyszerűbben:

$$\forall i(1 \le i \le N) \quad s(i) = s(i-1) + x(i), \quad s(0) = 0$$

Ezután egy intervallum elemei összege így fejezhető ki (1. ábra):

$$\sum_{i=a}^{b} x_i = s(b) - s(a-1)$$

Sorszámok	0	1	2	3	4	5	6	
Sorozat		3	5	1	8	2	4	
prefix sum	0	3	8	9	17	19	23	
sum(1,6)	0	3	8	9	17	19	23	
sum(1,2)	0	3	8	9	17	19	23	
sum(3,5)	0	3	8	9	17	19	23	19-8=11

1. ábra: prefix sum

Az alábbi algoritmus – a kumulatív összegzés módszere – így N^2-tel arányos lépést tesz meg.

```
Kumulatív összegzés(max,a,b):
  s[0]:=0
  Ciklus i=1-től N-ig
    s[i]:=s[i-1]+X[i]
  Ciklus vége
  max:=-∞
  Ciklus i=1-től N-ig
    Ciklus j=i-től N-ig
      Ha s[j]-s[i-1]>max akkor max:=s[j]-s[i-1]; a:=i; b:=j
    Ciklus vége
  Ciklus vége
Eljárás vége.
```

3. Kumulatív összegzés variációk

Ez a stratégia nem csak összegzéskor (sorozatszámításkor) használható, hanem más feladattípusoknál is. Alapvetően olyan esetekben, amikor a műveletek asszociatívak [7], van „inverze" [2], azaz

$$f(a,b) = f^{-1}\big(f(1,b), f(1,a-1)\big).$$

Például

$$\sum_{i=a}^{b} X_i = \sum_{i=1}^{b} X_i - \sum_{i=1}^{a-1} X_i$$

vagy

$$X_a * X_{a+1} * \ldots * X_b = \frac{X_1 * X_2 * \ldots * X_b}{X_1 * X_2 * \ldots * X_{a-1}}$$

feltéve, hogy egyik érték sem 0.

A maximumkiválasztás feladat `[a,b]` intervallumra azonban a fentiekre sem mindig érdekes, a szummánál például érdektelen, ha minden szám pozitív, a produktumnál pedig, ha minden szám nagyobb `1`-nél.

Sokszor azonban a feladat nem maximumkiválasztás, hanem sok kérdés megválaszolása különböző intervallumokra [8,9]. Egy másik típusban bizonyos típusú szummát keresünk, a [10] versenyfeladatban például a leghosszabb sorozatot, ahol a számok összege osztható `7`-tel.

A megszámolás tételre is lehet értelme, de akkor vagy az intervallum hosszát, vagy valami más tulajdonságot kell módosítani.

Ilyenek lehetnek:

- adjuk meg egy sorozat legfeljebb `K` hosszú részét, amelyben az adott tulajdonságú elemek száma maximális;
- adjuk meg egy sorozat legfeljebb `K` hosszú részét, amelyben az adott tulajdonságú elemek száma maximális és a két szélső elem is adott tulajdonságú;
- adjuk meg egy sorozat leghosszabb részét, amelyben az elemek legalább fele adott tulajdonságú és a két szélső elem!

Sok feladatban a módszer kiegészítőként szerepel, ráadásul csak bizonyos előfeltételek teljesülése esetén.

Feladat: Egy véletlenszám generátor `0` és `M-1` közötti számokat állít elő. Megkaptuk a generátor által előállított első `N` darab véletlenszámot. Írj programot, amely a generátor ellenőrzéséhez kiszámolja a sorozat `K` darab `[A,B]` intervallumára, hogy hány `0` és `M-1` közötti szám nem fordul elő az adott intervallumban! `K` értéke sokkal nagyobb, mint `M` értéke!

Írhatunk rá egy nem annyira naiv megoldást (indexeléssel oldja meg a számlálást):

```
Nincs(N,X,K):
  Ciklus i=1-től K-ig
    Be: A,B
    szam[]:=(0,...,0)
    Ciklus j=A-tól B-ig
      szam[X[j]]:=szam[X[j]]+1
    Ciklus vége
    C:=0
    Ciklus j=0-tól M-1-ig
      Ha szam[j]=0 akkor C:=C+1
    Ciklus vége
    Ki: C
  Ciklus vége
Eljárás vége.
```

A futási ideje ezek szerint `K* (N+M)` -mel arányos.

A kumulatív összegzés először kiszámolja egy mátrixban, hogy az első `i` véletlenszámból hány `j` értékű van, majd ebből számolja az eredményt:

```
Nincs(N,X,K):
  szam[0]:=(0,...,0)
  Ciklus i=1-től N-ig
    szam[i]:=szam[i-1]; szam[i,X[i]]:=szam[i,X[i]]+1
  Ciklus vége
  Ciklus i=1-től K-g
    Be: A,B
    C:=0
    Ciklus j=0-tól M-1-ig
      Ha szam[b,j]-szam[a-1,j]=0 akkor C:=C+1
    Ciklus vége
    Ki: C
  Ciklus vége
Eljárás vége.
```

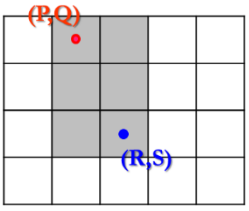
Az első ciklus a tömbértékadás lassú megvalósítása esetén is `N*M`-es, a második pedig `K*M`-es. Mivel a feladatleírás szerint `K` sokkal nagyobb `M`-nél, azért `K*N+K*M`, akkor nagyobb `N*M+K*M`-nél, és ez akkor igaz, ha `K>M`, mint a feladatleírásban szerepelt.

4. Kumulatív összegzés és matematika

A feladat nemcsak vektorokra, hanem más struktúrákra is általánosítható, például a következőben egy mátrixra [11]:

Feladat: Egy földműves egy téglalap alakú területet szeretne vásárolni egy `N×M`-es téglalap alakú földterületen. Tudja minden megvásárolható földdarabról, hogy azt megművelve mennyi lenne a haszna vagy vesztesége. Add meg azt a téglalapot, amelyen a legnagyobb haszon érhető el!

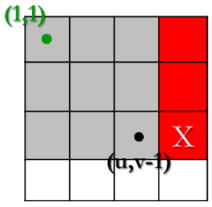
Az elemi megoldásban minden `(P,Q)` bal felső, `(R,S)` jobb alsó sarkú résztéglalapnak ki kellene számolni az elemei összegét, ami 6 ciklus eredményezne, majd ezek maximumát vehetnénk (2. ábra).



2. ábra: feladat

Próbáljunk valami részcélt kitűzni: számoljuk ki az `(1,1)` bal felső, `(u,v)` jobb alsó sarkú téglalapok értékét!

`X = a szürke téglalap értéke + a piros téglalap összege` (3. ábra)

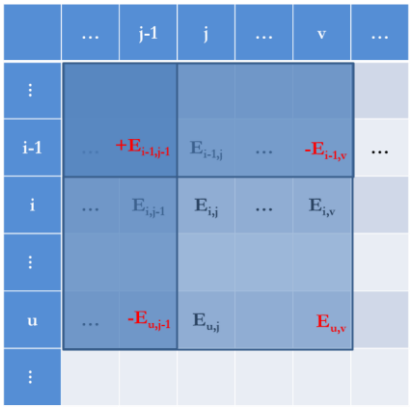


3. ábra: E kiszámítása

```
E[1..N,0]:=(0,...0)
Ciklus v=1-től M-ig
  x:=0
  Ciklus u=1-től N-ig
    x:=x+T[u,v]
  Ciklus vége
  E[u,v]:=E[u,v-1]+x
Ciklus vége
```

Az `(i,j)` bal felső, `(u,v)` jobb alsó sarkú téglalap értékei összege ezek alapján:

`érték(E,i,j,u,v)=E[u,v]-E[u,j-1]-E[i-1,v]+E[i-1,j-1]` (4. ábra)



4. ábra: az eredmény kiszámítása

Az algoritmust csak azért írjuk le, hogy a következő lépésben módosíthassuk:

```
max:=-∞
Ciklus i=1-től N-ig
  Ciklus j=1-től M-ig
    Ciklus k=i-től N-ig
      Ciklus l=j-től M-ig
        o:=érték(E,i,j,k,l)
        Ha o>max akkor P,Q,R,S:=i,j,k,l; max:=o
      Ciklus vége
    Ciklus vége
  Ciklus vége
Ciklus vége
```

Jól látható a 4 ciklus, azaz ha N és M azonos nagyságrendű, akkor a futási idő $O(N^4)$, azaz n negyedik hatványával arányos.

A versenyeken mindkét korosztálynak ugyanúgy módosítottuk a feladatot, rögzítettük a kiválasztandó téglalap területét, azaz olyan négy értéket kerestünk, amelyre egy újabb feltétel volt, hogy `(R-P+1)*(S-Q+1)=A`, egy adott A konstansra, illetve összeg helyett valamilyen típusú elemek számát kellett megadni.

A 2017/18 évben megrendezett verseny 3. fordulójának egyik feladata így szólt: „Egy téglalap alakú területen ismerjük `N*M` pont tengerszint feletti magasságát. Csúcsnak nevezzük azokat a pontokat, amelyek nagyobbak négy szomszédjuknál. A terület szélein biztos nincs csúcs. Készíts programot, amely megad egy pontosan A pontot tartalmazó téglalap alakú területet, amelyen a lehető legtöbb csúcs van!”

A versenyzők jelentős része, ahogy a korábbiak alapján várható volt, a naiv megoldást adta, 6 ciklussal. A módszert ismerők (kitalálók) jelentős része a fenti 4 ciklus belsejében vizsgálta a terület méretét:

```
    Ha (k-i+1)*(l-j+1)=A akkor
        o:=érték(E,i,j,k,l)
        Ha o>max akkor P,Q,R,S:=i,j,k,l; max:=o
```

Az ennél picit ügyesebbek még egy ciklust megspórolhattak volna és az `l` változó értékét csak olyan `k` értékekre számolták ki, amelyekre `(k-i+1)` osztotta A-t, de ilyen megoldást alig találtunk.

```
max:=-∞
Ciklus i=1-től N-ig
  Ciklus j=1-től M-ig
    Ciklus k=i-től N-ig
      Ha A mod (k-i+1)=0 akkor
        o:=érték(E,i,j,k,j+A div k-1)
        Ha o>max akkor P,Q,R,S:=i,j,k, j+A div k-1; max:=o
    Ciklus vége
  Ciklus vége
Ciklus vége
```

Az igazi megoldásban hatékonyan, `négyzetgyök(A)` [1] lépésben előre kiszámolhatók az `A` osztói, egy `C` elemű `D` vektorba, hiszen a terület két oldal szorzataként számolható.

[1] Elég az osztókat négyzetgyök(A)-ig meghatározni, hiszen ha x osztója A-nak, akkor A/x is osztója.

Így a várt megoldás $O(N^2C)$, kihasználva, hogy `D[k]*D[C-k+1]=A`:

```
max:=-∞
Ciklus i=1-től N-ig
  Ciklus j=1-től M-ig
    Ciklus k=1-től C-ig
      o:=érték(E,i,j,i+D[k]-1,j+D[C-k+1]-1)
      Ha o>max akkor P,Q,R,S:=i,j,i+D[k]-1,j+D[C-k+1]-1; max:=o
    Ciklus vége
  Ciklus vége
Ciklus vége
```

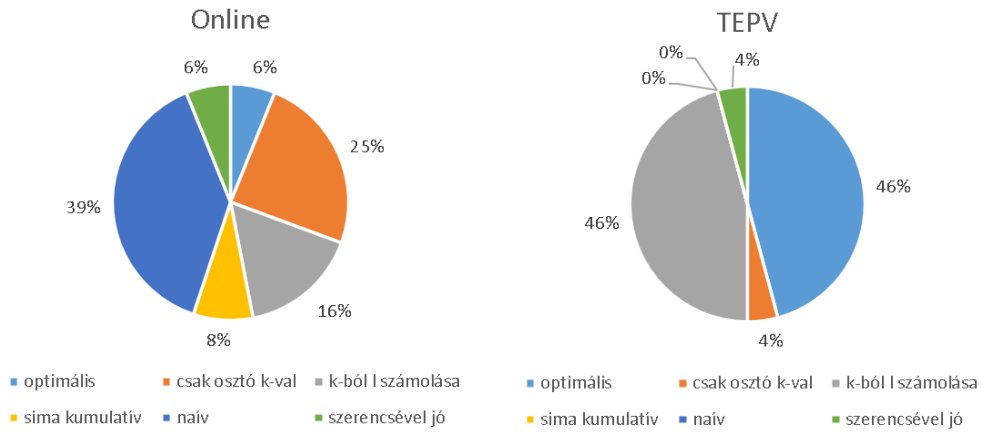
A tesztesetek alapján elértük, hogy a különböző megoldási módok eltérő pontszámot érjenek el, így a versenyzők által elért pontszámok alapján meghatározhatjuk megoldási típusok eloszlását (5. ábra).

Az egyes csoportokon belüli eltérő pontszám a speciális esetek jó vagy rossz kezelése miatt lehet.

optimális, csak osztó `k`-val, `k` -ból `l` számolása	92-100
csak osztó `k`-val, csak osztó `l`-lel	71-91
`k`-ból `l` számolása	55-70
sima kumulatív	41-54
naiv	16-40
szerencsével jó	0-15

5. ábra: megoldási módok ponthatárai

Látszik, hogy az egyetemisták a több matematikai ismeretüket felhasználva több pontot érnek el, illetve a naiv és a kumulatív megoldásból már tovább léptnek egy hatékonyabb megoldás felé. Náluk naiv és sima kumulatív megoldás nem is volt. Ezzel szemben ezek együtt a 13-16 éves korosztálynál majdnem elérték az 50%-os arányt. A legjobbak azonban közülük is megtalálták az optimális megoldást (6. ábra).



6. ábra: A megoldástípusok eloszlása

6. Konklúzió

A kumulatív összegzés alapgondolata tehát: Tetszőleges részsorozaton számolt érték helyett csak olyan részsorozatokra számoljunk értéket, amelyek kezdete az egész sorozat kezdete, majd a részsorozaton számolt értéket fejezzük ki ezekből!

Összegzés helyett megszámolási feladatokra is alkalmazható ugyanez az elv, azaz a művelet bármely asszociatív művelet lehet, amelynek számolható az inverze.

A verseny tapasztalataink alapján a diákok az alapmódszert könnyen megtanulják, de maguktól nem találják ki – jellemzően minél fiatalabbakat nézünk, annál inkább nő a naiv, lassú megoldást készítők aránya. A pici módosításokat csak egy részük veszi észre, pl. amikor elölről és hátulról is kell számolni ilyen összegeket, vagy összegzés helyett más műveletet kell használni.

Fontos gondolat, hogy a naiv megoldások a legtöbb feladat esetén összegzést és maximumkiválasztást tartalmaznak, de a hatékony, kumulatív összegzést alkalmazók megírásához is ezek az elemi algoritmusok (programozási tételek) kellenek. Azaz nem új algoritmusokról van szó, hanem a megszokott algoritmusok alkalmazásában kell másképp gondolkodnunk!

A matematikai meggondolások, mint az utolsó feladatban láthattuk, azonban a matematikából nem erős diákoknak problémát okoznak. A matematika pedig csak annyi volt, hogy ha egy egész szám két egész szám szorzataként áll elő, akkor mindkét számnak osztania kell a szorzatot – tehát szükség van az osztók előállítására. Az eredményekből is látható, hogy az első félévre járó egyetemisták lényegesen nagyobb arányban oldották meg a feladatot maximális pontszámmal.

A fentiek miatt úgy gondoljuk, hogy a programozás oktatásában a fenti alapmódszerrel (minden bonyolultabb adatstruktúrát kerülve) mindenképpen foglalkoznunk kell. Ebben a tanárok jelenleg kevés segítséget kapnak [12], mert a tudományos cikkek elsöprő többsége a különböző fákat használó fejlett megoldásokról szól.

Irodalom

1. J. Bentley: *Programming Pearls - Algorithm Design Techniques.* In: Comm. ACM (1984), Vol 27. Issue 9, pp. 865-871
2. Johan Sannemo: *Principles of Algorithmic Problem Solving.* https://usaco.guide/PAPS.pdf#page=207, 2018 (utoljára megtekintve: 2021.11.15.)
3. ACM; https://helloacm.com/category/teaching-kids-programming/ (utoljára megtekintve: 2021.11.15.)
4. E. S. Page: *Continuous Inspection Schemes. Biometrika.* Vol. 41. No. 1/2 (Jun., 1954), pp. 100-115, https://www.jstor.org/stable/2333009?origin=JSTOR-pdf&seq=1#metadata_info_tab_contents (utoljára megtekintve: 2021.11.15.)
5. Nemes Tihamér Online Programozási Verseny; http://tehetseg.inf.elte.hu/nemes-online/index.html (utoljára megtekintve: 2021.11.15.)
6. Tehetségkutató Egyetemi Programozási Verseny; http://versenyek.inf.elte.hu/versenyek/tehetsegkutato-egyetemi-programozasi-verseny/ (utoljára megtekintve: 2021.11.15.)
7. Guy E. Blelloch: *Prefix Sums and Their Applications. School of Computer Science*, Carnegie Mellon University, Pittsburgh, https://www.cs.cmu.edu/~guyb/papers/Ble93.pdf (utoljára megtekintve: 2021.11.15.)
8. Darren Yao, Dustin Miaoh: *Introduction to Prefix Sums.* https://usaco.guide/silver/prefix-sums?lang=cpp (utoljára megtekintve: 2021.11.15.)
9. USACO 2015 December Contest, Silver, *Problem 3. Breed Counting.* http://www.usaco.org/index.php?page=viewproblem2&cpid=572 (utoljára megtekintve: 2021.11.15.)

10. USACO 2016 January Contest, Silver Problem 2. Subsequences Summing to Sevens. http://www.usaco.org/index.php?page=viewproblem2&cpid=595 (utoljára megtekintve: 2021.11.15.)
11. Steven Halim: Classical problems. https://acm.wustl.edu/cse232/classical.md (utoljára megtekintve: 2021.11.15.)
12. Szlávi P., Zsakó L.: Elemi algoritmusok (a programozási tételek után). ELTE Informatikai Kar, Budapest. http://tehetseg.inf.elte.hu/szakkorefop2017/pdf/elteikszakkor_elemi_algoritmusok.pdf (utoljára megtekintve: 2021.11.15.)