

# Algel V. gyakorlat

## Rendezettek vagyunk

2012. március 5.

6. [ZH: 2007. április 27.] A valós számokból álló  $a_1^2 \dots a_n^2$  sorozat egy darabig nő, utána csökken. Adjon  $O(n)$  összehasonlítást használó algoritmust, ami rendezi az  $a_1, \dots, a_n$  sorozat!

Először meg kell keresni a töréspontot, ami  $O(n)$ -ben megvan. Az előtte lévő sorozatban a pozitívok sorozata nő, a negatívoké csökken, az utána lévőben pont fordítva. Így a két pozitív részt (a hátsót természetesen megfordítva)  $O(n)$ -ben össze tudjuk fésülni, hasonlóan a két negatívot is. A végén egyszerűen össze kell őket ragasztani. Költség: pesszimistán is  $4O(n)$ , ami  $O(n)$ .

7. Az  $A[1 : n]$  tömbben egy rendezett univerzum  $n$  különböző eleme volt, nagyság szerint növekvő sorrendben. Valaki időközben megkeverte a tömb elemeit, de csak annyira, hogy minden egyes elem új helye az eredetitől legfeljebb 5 távolságra esik. Adjunk  $O(n)$  futásidőjű algoritmust az eredeti állapot helyreállítására!

Egy 5 hosszú ablakot tolunk végig a sorozaton. Az első elem biztos, hogy az első 5 elem között van, így közöttük minimumot keresünk, így visszavezettük a feladatot egy  $n - 1$  méretű ugyanilyen feladatra. 5 elem közül 4 összehasonlítással tudunk minimumot keresni, ezt  $n$ -szer kell megtenni (a vége miatt kicsit kevesebbszer, de ez mindegy), így a lépésszám  $O(4n) = O(n)$ .

8. [ZH: 2010. április 19.] Az  $A$  tömbben  $n$  különböző számot tárolunk. Tudjuk, hogy  $A[1] > A[2]$  és  $A[n - 1] < A[n]$ . Adjunk algoritmust, mely  $O(\log n)$  összehasonlítással megtalál a tömbben egy lokális minimumot (ha van), azaz egy olyan  $1 \leq i \leq n$  indexet, hogy  $A[i]$  tömbbeli szomszédai nagyobbak, mint  $A[i]$ .

A feltételekből következik, hogy biztos van lokális minimum. Bináris keresést használunk, ha az adott körben  $A[i]$  egy lokális minimum, akkor kész vagyunk, ha  $A[i - 1] < A[i]$ , akkor a jobb oldalt dobjuk el, egyébként a balt. Tehát egy lépésben vagy megtaláltuk a kérdéses elemet, vagy kaptunk egy feleakkora tömböt, ami ugyanolyan tulajdonságú, mint az eredeti. Ez így  $O(\log n)$  a bináris keresés elve miatt.

9. [Vizsga: 2007. június 19.] Adott a síkon  $n$  pont, melyek koordinátái  $(a_1, b_1) \dots (a_n, b_n)$ . Olyan  $P = (x, y)$  pontot keresünk a síkon, amire az alábbi összeg minimális.

$$\sum_{i=1}^n (|a_i - x| + |b_i - y|)$$

Adjunk algoritmust, ami  $O(n \log n)$  lépésben meghatároz egy ilyen  $P$  pontot.

Észrevesszük, hogy  $x$  és  $y$  koordináta független. Egy dimenzióban: ha csak két pont van, akkor két pont között bárhol lehet, rajtuk kívül viszont rosszabb. Tehát a két szélső pont között kell lennie, de rajtuk kívül a két szélső között is stb, tehát páratlan pont esetén a középsőn, páros esetén a középső kettő közül valamelyiken vagy közöttük (indoklás: ha nem így lenne, akkor biztos, hogy tudnánk jobbat csinálni). Tehát koordinátánként egy rendezés, és középső választása, tehát  $O(n \log n)$ -ben bent vagyunk.

10. Legyen adott egy egészekből álló  $A[1 : n]$  tömb valamint egy  $b$  egész szám. Szeretnénk hatékonyan eldönteni, hogy van-e két olyan  $i, j \in \{1, \dots, n\}$  index, melyekre  $A[i] + A[j] = b$ . Oldjuk meg ezt a feladatot  $O(n \log n)$  időben!

A tömböt  $O(n \log n)$ -ben tudjuk rendezni. Triviális megoldásként tehetjük azt, hogy  $\forall i$ -re megkeressük bináris kereséssel, hogy szerepel-e a tömbben  $b - A[i]$  érték. Ha igen, kész vagyunk, ha egyszer sem találtunk ilyet, akkor pedig nem. Ez legfeljebb  $n$  db bináris keresés, azaz ez a része is megvan  $O(n \log n)$  lépésből. (Persze egy egyszerű dinamikus programmal a második részt lineáris időben is megoldhatjuk.)

11. [Vizsga: 2009. június 11.] Adott a számegyenesen  $n$  intervallum,  $[a_1, b_1], \dots, [a_n, b_n]$ . Azt akarjuk tudni, hogy együtt milyen hosszú részt fednek le a számegyenesből (azaz, hogy mennyi az  $\cup_{i=1}^n [a_i, b_i]$  összhossza). Adjon  $O(n \log n)$  lépéses algoritmust ennek a hosszának a meghatározására!
- Rendezzük az intervallumokat növekvő sorrendbe az  $a_i$  koordináta szerint! A következő dolgokat tartjuk nyilván:  $\min_a, \max_b, \text{sum}$ . Kezdetben  $\min_a = a_1, \max_b = b_1$ .
- Ha a sorban  $[a_i, b_i]$  intervallum következik: ha  $a_i < \max_b$ , akkor még a vizsgált intervallumban vagyunk, így  $\max_b = \max(\max_b, b_i)$ . Ha  $a_i > \max_b$ , akkor a következő intervallumig biztos van szünet. Ekkor ezt tesszük:  $\text{sum}+ = \max_b - \min_a, \min_a = a_i, \max_b = b_i$ . A legutolsó intervallum feldolgozása után (az utolsó nyitott intervallum-jelölt lezárása miatt)  $\text{sum}+ = \max_b - \min_a$ .
- Bizonyítás pl. indukcióval (lényegében ez egy dinamikus programozásos feladat is egyben). Lépésszám: a tényleges számolás  $O(2n)$ -ben megvan, így a rendezéssel együtt a lépésszám  $O(n \log n)$ .
12. Adottak a sík egész koordinátájú  $P_1 = (x_1, y_1), \dots, P_n = (x_n, y_n)$  koordinátájú pontjai. Javasoljunk  $O(n)$  költségű módszert olyan  $P_i \neq P_j$  pontok kiválasztására, amelyeken átmenő egyenes által meghatározott félsíkok közül az egyik tartalmazza az összes pontot!
- Minimális  $y$  koordinátájú pont lesz az egyik, ami meghatározza, a másik pedig az  $y_{\min}$ -es pontból abszolútértékben legnagyobb meredekségű. Ha ezeken kívül esne egy pont, akkor annak a meredeksége abszolútértékben nagyobb lenne. Meredekséget számolni két pont ismeretében konstans, így a lépésszám két minimumkeresés, azaz  $O(n + n) = O(n)$ . (Persze ugyanezt lehet  $x$  koordináta szerint is, valamint minimum helyett maximummal is.)
13. [Vizsga: 2004. június 10.] Az  $n$  méretű (nem feltétlenül rendezett)  $A$  tömb elemei különböző pozitív egész számok. Adjon algoritmust, amely meghatároz egy  $1 \leq k \leq n$  számot és kiválaszt  $k$  különböző elemet az  $A$  tömbből úgy, hogy a kiválasztott elemek összege nem több, mint  $k^3$ . Ha nincs ilyen  $k$ , akkor az algoritmus jelezze ezt a tényt! Az algoritmus lépésszáma legyen  $O(n \log n)$ ! (Két szám összehasonlítása, összedáása vagy szorzása egy lépésnek számít.)
- Észrevétel: ha van ilyen  $k$ , akkor ezeket kicserélhetjük a  $k$  legkisebbre, hiszen az összegük így csak csökkenhet, ami továbbra is jó. Tehát elég mindig a  $k$  legkisebbel foglalkozni. Rendezzük a tömböt, és induljunk el az elejétől. Folyamatosan összegezzük, és az első  $k$ -ra ahol igaz, kész is vagyunk, ha pedig végigértünk, és nem volt ilyen, akkor biztos nincs. Ez így rendezés és egy végigolvasás, azaz  $O(n \log n + n) = O(n \log n)$ .
14. [Vizsga: 2004. június 3.] A  $2^k - 1$  elemű  $A$  tömb elemei mind különbözőek és növekvő sorrendben vannak. Minden elemet egy  $k$  hosszú bitsorozat ír le, tehát tekinthetjük úgy, hogy a  $0, 1, 2, \dots, 2^k - 1$  számokat tároljuk egy kivételével. A feladat ennek a hiányzó számnak a megkeresése. Ehhez egy lépésben valamelyik elem egy bitjére kérdezhetünk rá: a  $BIT(i, j)$  eljárás az  $A[i]$  elem  $j$ -edik bitjét mondja meg. Adjon olyan algoritmust, amely a  $BIT$  eljárás  $O(k)$ -szori hívásával megtalálja a hiányzó számot (bitsorozatot).
- Mivel egy hiányzik, egy helyen elromlik a paritás (aminek ellenőrzéséhez elég az utolsó számjegyre rákérdezni). Ezt a helyet keressük meg bináris kereséssel, ami  $O(\log(2^k - 1)) = O(k)$ .
15. Adott egy dobozban  $n$  különböző méretű anyacsavar, valamint egy másik dobozban a hozzájuk illő apacsavarok. Kizárólag a következő összehasonlítási lehetőségünk van: egy apacsavarhoz hozzápróbálunk egy anyacsavart. A próbának háromféle kimenete lehet:  $\text{apa} < \text{anya}$ ,  $\text{apa} = \text{anya}$ ,  $\text{apa} > \text{anya}$ , annak megfelelően, hogy az apacsavar külső átmérője hogyan viszonyul az anyacsavar belső átmérőjéhez. Szeretnénk minden anyacsavarhoz megtalálni a megfelelő apacsavart. Adjunk erre a feladatra átlagosan  $O(n \log n)$  összehasonlítást felhasználó módszert!
- A gyorsrendezés pontosan ilyen műveleteket végez, az pont jó lesz itt is.

16. **Igazoljuk, hogy egy  $n$  elemből álló kupac felépítése  $\Omega(n)$  összehasonlítást igényel!**  
 Fejet asztalbaverően egyszerű bizonyítás: mivel a gyökérben a legkisebb elem van, ezért nem lehet hamarabb felépíteni, mint a legkisebb elemet megtalálni, ami pedig önmagában is  $\Omega(n)$ .

17. **\*[Vizsga: 2003. május 30.] Adott összesen  $2n$  különböző szám két  $n$  elemű halmazban,  $A = \{a_1, \dots, a_n\}$  és  $B = \{b_1, \dots, b_n\}$ . Azt szeretnénk eldönteni minimális számú összehasonlítással, hogy a  $2n$  szám közül a legnagyobb az  $A$  vagy a  $B$  halmazban van-e. (Azaz nem kell feltétlenül meghatározni, melyik elem a legnagyobb, csak azt, hogy melyik halmazba tartozik.) Mutassa meg, hogy ehhez a feladathoz is legalább annyi összehasonlítás kell, amennyi  $2n$  elem közül a maximális meghatározásához szükséges.**

Tfh van olyan algoritmus, ami legfeljebb  $2n-2$  öh-t használ. Ekkor adjunk neki  $2n$  elemet, és mi leszünk az ellenség. Jelöljük meg magunknak az összes elemet, mert az eddigi 0 kérdés alapján bármelyik lehet a maximális. Tartsuk továbbá azt is nyilván, hogy melyik halmazban mennyi elem lehet globálisan maximális (kezdetben  $n$  és  $n$ ). Ha a feltételezett algoritmus rákérdez két elemre, akkor a következőképpen mondjuk meg a viszonyukat:

- $a_i ? b_j$ , az eddigi válaszainkból már kiderült a viszonyuk: megmondjuk az igazságot (a lehetséges maximális elemek halmaza ilyenkor nem változik)
- $a_i ? a_j$ : ha már kiderült a viszonyuk, akkor megmondjuk, ha nem, akkor tetszőleges viszonyt mondunk rájuk (ezt feljegyezzük), és kihúzzuk a kisebbet a jelöltek közül, meg a számlálót is megfelelően változtatjuk
- $b_i ? b_j$ : mint előbb
- $a_i ? b_j$ , az eddigi válaszainkból még nem derül ki a viszonyuk: eldönthetjük, hogy melyik legyen a kisebb. Ebben az esetben pontosan 1 elem maximalitása lesz kizárva. Úgy választjuk meg a relációjukat, hogy abból a halmazból zárjuk ki valakinek a maximalitását, amelyikben a több jelölt van. Ezután megcsináljuk a szükséges adminisztrációt (kihúzzunk egy jelöltet és a számot aktualizáljuk).

Meg kell gondolni, hogy az így definiált relációkhoz mindig tudunk számokat kitalálni (a későbbi anyagban lesz erről részletesebben szó, lásd topologikus rendezés). Ezen túl mivel egy kérdéssel legfeljebb 1 elem maximalitása zárható ki, így  $2n - 2$  kérdés után még legalább 2 olyan elem van, ami lehet maximális ( $\equiv$  megválasztható maximálisnak úgy, hogy az eddigi válaszainkkal konzisztens legyen). Viszont a stratégiánk miatt biztos, hogy mindkét halmaz tartalmaz ilyen, így ennyi kérdés biztos nem elég a feladat megoldásához.

18. **\*[ZH: 2002. április 8.] Adottak a  $c_1, c_2, \dots, c_n$  különböző egész számok. Ezeket szeretnénk nagyság szerint rendezni növekvő, vagy csökkenő sorrendbe úgy, hogy a szokásos összehasonlítás helyett, most a következő kérdéseket lehet feltenni: *Három kiválasztott elem közül melyik a középső?* Bizonyítsuk be, hogy a leghatékonyabb algoritmus  $\Theta(n \log_2 n)$  összehasonlítást használ!**

$\Omega(n \log_2 n)$ : ugyanúgy kell bizonyítani, mint az öh. alapú rendezések alsó korlátját, csak itt kettő helyett három lehetőségünk van, így  $\Omega(\log_3 n!)$  fog kijönni. Ez viszont nem baj, mert  $\Omega(\log_3 n!) = \Omega(\log_2 n!) \approx \Omega(n \log_2 n)$ .

$O(n \log_2 n)$  (vázlatosan):  $O(n)$  lépésből tudunk szélsőértéket keresni: veszünk 3 elemet véletlenszerűen, a középsőt kidobjuk. Ezt megcsináljuk addig, amíg csak 2 marad, ez lesz a minimális és maximális (nem tudjuk, melyik melyik). Az egyiket válasszuk ki, ha ez  $x$ , akkor a  $kozepso(x, a, b)$  vagy az  $a < b$  kérdésre válaszol, vagy az  $a > b$  kérdésre (attól függ, hogy a minimálisat vagy maximálisat választottuk, és egy adott  $x$ -re ez konzisztens). Azaz az így implementált  $<$  függvény tényleg egy rendezést (vagy növekvő, vagy csökkenő; nem tudjuk) definiál a számainkon:

$$< (a, b) \{ \text{return } kozepso(x, a, b) == a \}$$

Ez konstans időben számolható, tehát bármely  $<$  operátort használó rendezés használható lesz ezzel a trükkel, abból pedig van  $O(n \log_2 n)$  lépésszáma.