Algel V. gyakorlat

Rendezzük sorainkat

2011. március 7.

8. **[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 $O(n)$.

9. **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 $O(n)$ futásidejű 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 $O(4n) = O(n)$.

10. **[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.**

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

    Adjon 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.

11. **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.)

12. **[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 hossznak a meghatározására!**
    Rendezzük az intervallumokat növekvő sorrendbe az a_i koordináta szerint! A következő dolgokat tartjuk nyilván: $\min_a$, $\max_b$, *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: $sum+ = \max_b - \min_a$, $\min_a = a_i$, $\max_b = b_i$. A legutolsó intervallum feldolgozása után (az utolsó nyitott intervallum-jelölt lezárása miatt) $sum+ = \max_b - \min_a$. Bizonyítás pl. indukcióval (lényegében ez egy dinamikus programozásos feladat is egyben). Lépésszám: a tényleges számolás $O(2n)$-ben megvan, így a renezéssel együtt a lépésszám $O(n \log n)$.

13. **Adottak a sík egész koordinátájú $P_1 = (x_1, y_1), \ldots, P_n = (x_n, y_n)$ 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 kooridántájú 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.)

14. **A (növekvően) rendezett $A[1 : n]$ tömb k elemét valaki megváltoztatta. A változtatások helyeit nem ismerjük. Javasoljunk $O(n + k \log k)$ költségű algoritmust az így módosított tömb rendezésére!**
Menjünk végig a tömbön, és a hibás párokat (amik rosszul állnak) szedjük külön. Ez azért kell, mert a rossz szomszédok közül az legalább az egyiket biztos megváltoztattuk (hiba csak változtatásnál lehet), viszont nem tudjuk, melyiket. Így legfeljebb $2k$ elemet vettünk ki, ezeket $O(2k \log 2k = O(k \log k)$ időben tudjuk rendezni, utána $O(n+k)$ időben összefésülés. A kezdeti végigolvasás $O(n)$, vagyis összességében tényleg $O(n + k \log k)$ lépésben végzünk.

15. **Vázoljunk egy $O(n)$ időigényű algoritmust (az időkorlát bizonyításával együtt) n olyan egész számból álló sorozat rendezésére, melynek elemei az**

    (a) **$\{1, \ldots, 3n\}$ tartományba esnek!**
    Ládarendezés, $3n$ ládával, $O(n + 3n) = O(n)$.

    (b) **$\{1, \ldots, n^7 - 1\}$ tartományba esnek!**
    n-es számrendszerben felírjuk a számokat (ez darabonként 7 osztás, vagyis konstans), utána számjegyenként radix (legfeljebb 7 jegyű számok), azaz $O(7n + 7n) = O(n)$.

16. **[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, összadása 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 $O(n \log n + n) = O(n \log n)$.

17. **[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 $0, 1, 2, \ldots, 2^k - 1$ 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 $O(\log(2^k - 1)) = O(k)$.

18. **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: apa<anya, apa=anya, apa>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.

19. **A 4 elemű I abc felett adott két szó: $x = x_1 x_2 \dots x_n$ és $y = y_1 y_2 \dots y_k$, ahol $1 \leq k \leq n$ és $\forall i, j : x_i, y_j \in I$. Keressük az x szóban az olyan részszavakat, amelyek anagrammái y-nak, azaz az olyan i indexeket, hogy az $x_i, x_{i+1}, \dots, x_{i+k-1}$ betűk megfelelő sorrendbe rakva az y szót adják. Adjunk algoritmust, ami x-ben az összes ilyen i helyet $O(n)$ lépésben meghatározza!**
Csináljunk egy hisztogramot y-ból, vagyis egy 4 elemű tömbbe írjuk be, hogy melyik karakterből hányat tartalmaz (magyarul ládarendezés). Ez $O(k) = O(n)$. Ezután csináljunk egy hasonló tömböt x első k elemére (T_x), ami szintén ugyanennyi idő. Ha a két tömb megegyezik (ezt a 4 elem miatt $O(4) = O(1)$-ben tudjuk ellenőrizni), akkor találtunk egy anagrammát. Ha az ablakot (ami általánosan az x_i-nél kezdődik) eggyel jobbra mozgatjuk, akkor $--T_x[x_i]$ és $++T_x[x_i + k + 1]$ (azaz az elhagyott karaktert kivesszük, a következőt betesszük). A helyesség könnyen látható, lépésszám: mind az n pozícióra (igazából kicsit kevesebbre) két tömbművelet és egy 4 hosszú egyenlőségvizsgálat (azaz összességében konstans), tehát $O(n)$-ben ez a része is megvan.

20. **[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:

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

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