Dinamikus programozás, rendezések eleje

1. Adott n pozitív egész szám, $a_1, a_2, \ldots, a_n$. Az $n+1$ sorból és $b+1$ oszlopból álló T táblázat sorait 0-tól n-ig, oszlopait 0-tól b-ig indexeljük. Legyen $T[0,0] = 1$ és $T[0,c] = 0$ minden $1 \leq c \leq b$ értékre. Adjon eljárást, ami a T többi mezőjét összesen $O(nb)$ lépés alatt kitölti úgy, hogy $T[i,c]$ értéke azt mutassa, hányféleképpen lehet az $a_1, a_2, \ldots, a_i$ számok közül néhány összegeként a c számot előállítani ($1 \leq i \leq n, 1 \leq c \leq b$).

   Megoldás: Vegyük észre, hogy a megadott nulladik sor is megfelel az értelmezésnek: a 0 hosszú sorozatból tetszőleges $c > 0$ szám csak nulla féleképpen állítható elő.

   A $c = 0$ akármilyen hosszú pozitív sorozatból egyféleképpen áll elő: úgy, hogy egyik számot se választjuk ki (üres összeg). Ennek megfelelően a nulladik oszlop végig 1 lesz.

   Nézzük az $i \geq 1$, $c \geq 1$ eseteket! Az első i számból a c-t vagy úgy állíthatjuk elő, hogy az a_i számot nem is használjuk, ebből van $T[i-1,c]$ lehetőség, vagy ha $c \geq a_i$, akkor az a_i számot is használhatjuk, ekkor a korábbiakból a maradék $c - a_i$ összeget kell előállítani, tehát: $T[i,c] = T[i-1,c] + T[i-1,c-a_i]$ ha $c \geq a_i$ és $T[i,c] = T[i-1,c]$ ha $c < a_i$.

   A táblázat kitöltése történhet pl. soronként, azaz minden $i = 1, \ldots, n$ értékre kitöltjük a $c = 0, 1, \ldots, b$ mezőket.

   A táblázat mérete $(n+1) \cdot (b+1) \in O(nb)$. Minden elemének kitöltése konstans lépés, tehát az összes idő $O(nb)$.

2. Tekintsük az RH problémának azt a változatát, amikor adottak az $a_1, a_2, \ldots, a_n$ és a b egész számok, melyekre teljesül, hogy $0 < a_i < n^2$ minden $1 \leq i \leq n$ esetén. Kérdés, hogy van-e olyan $I \subseteq \{1, \ldots, n\}$, melyre $\sum_{i \in I} a_i = b$. Mutassa meg, hogy ez a változat P-ben van!

   Megoldás: Vegyük észre, hogy ha $b > a_1 + a_2 + \cdots + a_n$, akkor biztos nincs megoldás, tehát a $b \geq n^3$ esetben az algoritmus térjen vissza a ,,nem" válasszal. Ha viszont $b < n^3$, akkor elkészíthetjük lényegében az előző feladatbeli táblázatot (nem kell feltétlenül számon tartani, mi hányféleképpen áll elő, elég csak az előállás tényét rögzíteni, azaz használhatjuk a $T[i,c] = T[i-1,c] \vee T[i-1,c-a_i]$ rekurziót is). Azt már tudjuk, hogy ez a táblázat $O(nb)$ időben kitölthető, de ezt most a b-re alkalmazott korlát miatt $O(n^4)$ időben tudjuk kitölteni, tehát ez az eljárás polinomiális.

3. Adjon $O(n^2)$ lépésszámú, dinamikus programozást használó algoritmust, ami megtalálja egy n hosszú $a_1, a_2, \ldots, a_n$ számsorozatban a leghosszabb növekvő részsorozatot. Például a $10, 3, 5, 2, 7, 1, 18, 4, 12, 17, 6$ sorozatban a leghosszabb növekvő részsorozat a $3, 5, 7, 12, 17$.

   Megoldás:

   (a) Részfeladatok: minden $1 \leq i \leq n$-re $M[i]$ legyen a leghosszabb olyan növekvő részsorozat hossza, aminek utolsó tagja a_i.

   (b) Haladjunk $i = 1,2, \ldots, n$ sorrendben.

   (c) $M[1] = 1$ hiszen a_1 magában jó, ennél több meg nem lehet, mert nincs a_1 előtt semmi.

   (d) $M[i] = 1 + max\{M[j] \mid j < i$ és $a_j < a_i\}$, ha van legalább egy olyan a_j, amire $j < i, a_j < a_i$, kben $M[i] = 1$
   Ez ezért jó, mert a_i-t mindenképp választanunk kell és a leghosszabb itt végződő monoton részsorozat egy olyan korábbi a_j-ben végződő leghosszabb sorozat folytatása, amire $a_j < a_i$, ha van ilyen a_j, különben pedig a_i önmagában állva alkotja a legjobb ilyen sorozatot.

   (e) A leghosszabb sorozat valahol végződik, $max\{M[i] \mid 1 \leq i \leq n\}$, adja meg a leghosszabb növekvő részsorozat hosszát.

   Ahol az 5. pontban a maximum felvevődik, ott lesz a vége a legjobb részsorozatnak és ha azt is feljegyezzük az $M[i]$-k kiszámoláskor, hogy melyik j-nél volt a maximum (mi az eggyel korábbi érték a sorozatban), akkor a végén visszakövethető maga a sorozat is.

   Az egész eljárás lépésszáma azért $O(n^2)$, mert n feladatot oldunk meg és mindegyikhez legfeljebb $O(n)$ esetet kell megnéznünk, legfeljebb n korábbi a_j van.

4. Az $A[1:n]$ tömbben levő elemekről tudjuk, hogy $A[1] \neq A[n]$. Adjon $O(\log n)$ összehasonlítást használó algoritmust, amely talál egy olyan i indexet, hogy $A[i] \neq A[i+1]$.

   Megoldás: Bináris keresést használunk: legyen $i = \lceil n/2 \rceil$. Ha $A[i] \neq A[1]$, akkor az $A[1:i]$ tömb is ugyanolyan tulajdonságú, mint az eredeti, elég ebben keresni. Ha viszont $A[i] = A[1]$, akkor nyilván $A[i] \neq A[n]$, és akkor a továbbiakban az $A[i:n]$ tömbben kereshetünk.

   Az eljárás végén a tömb leszűkül egy 2 eleműre, azaz két szomszédos nem egyenlő elemre, ekkor készen vagyunk.

   A lépésszám, ahogy a bináris keresésnél $O(\log n)$.

5. Rendezze a $3, 12, 1, 34, 4, 6, 0$ számsorozatot beszúrásos rendezéssel! Hány összehasonlításra volt szükség?

   Megoldás: A beszúrásos rendezésnek két fajtája van: amikor lineárisan és amikor binárisan keresünk. A beszúrásonként kapott sorozatok ugyanazok, de az összehasonlítások száma eltérhet.

   Az eljárás az, hogy az i-edik elemet szúrjuk be a már rendezett első $i-1$ elem közé, tehát a sorozat az egyes lépésekben így alakul: $(3, \mathbf{12}, 1, 34, 4, 6, 0) \longrightarrow (3, 12, \mathbf{1}, 34, 4, 6, 0) \longrightarrow (1, 3, 12, \mathbf{34}, 4, 6, 0) \longrightarrow (1, 3, 12, 34, \mathbf{4}, 6, 0) \longrightarrow (1, 3, 4, 12, 34, \mathbf{6}, 0) \longrightarrow (1, 3, 4, 6, 12, 34, \mathbf{0}) \longrightarrow (0, 1, 3, 4, 6, 12, 34)$.

   Az összehasonlítások száma lineáris keresésnél (úgy hangzott el, hogy előről keresünk): 1+1+3+3+4+1 = 13, bináris keresésnél: $1 + 1 + 2 + 2 + 2 + 2 = 10$.

6. Tudjuk, hogy az $a_1, \ldots, a_n$ sorozat olyan, hogy egy darabig növekszik, utána csökken. Adjon $O(n)$ összehasonlítást használó algoritmust, ami növekvő sorrendbe rendezi az elemeit!

   Megoldás: Ötlet: a legnagyobb elemnél kettévágjuk a sorozatot. A kapott két rész mindegyike rendezett, ezeket össze tudjuk fésülni egyetlen rendezett sorozattá.

   Megvalósítás: sorban az $a_i - a_{i+1}$ szomszédokat összehasonlítva megtalálhatjuk a legnagyobb elemet, legyen ez a_x. Eddig $n-1$ összehasonlítást használunk. Az $a_1, \ldots, a_x$ és az $a_n, a_{n-1}, \ldots, a_{x+1}$ növekvő sorozatokat a tanult módon legfeljebb $n-1$ összehasonlítással összefésüljük, és így legfeljebb $2n-2 \in O(n)$ összehasonlítással készen vagyunk. (Az algoritmus mozgatja is az elemeket, de összesen ez is $O(n)$ lépés.)

   Megoldhatjuk az a_x előzetes megkeresése nélkül is. Elég azt képzelni, hogy két sorozatunk van, az egyik első eleme a_1, a másiké a_n (és kezdetben nem tudjuk, hol érnek véget). Legyen $i = 1$ és a $j = n$, és hasonlítsuk össze az a_i és az a_j elemet. A kisebb a készülő rendezett lista következő eleme, és ha $a_i < a_j$, akkor az i értékét növeljük, különben a j értékét csökkentjük. A továbbiakban is így, az összefésüléshez hasonlóan folytatjuk. Akkor lesz vége, ha $i = j$ (és akkor ez a legnagyobb elem). Ez az eljárás legfeljebb $n-1 \in O(n)$ összehasonlítást használ.

   Megjegyzés: mindkét eljárás helyesen működik abban az esetben is, amikor a legnagyobb elem az a_1 vagy az a_n, akkor eredetileg is egy rendezett sorozatunk van (növekvő vagy csökkenő).

7. Az eredetileg növekvő $a_1, \ldots, a_n$ sorozatban egy elem értéke megváltozott, de nem tudjuk melyik. Hogyan lehet $O(n)$ lépésben újra növekvő sorrendbe rendezni az elemeket?

   Megoldás: Vegyük észre, hogy ha a változtatás után is $a_i < a_{i+1}$ minden i-re, akkor bár nem tudjuk megállapítani, melyik változott, de erre nincs is szükség, a sorozat továbbra is rendezett.

   Ha viszont van olyan i, melyre $a_i > a_{i+1}$, akkor ez úgy keletkezett, hogy a_i és a_{i+1} valamelyike változott, a_i nőtt vagy a_{i+1} csökkent. legegyszerűb (nem feltételnül leggyorsabb) módszer, ha ezt a két problémás elemet beszúrjuk a többi elem által alkotott növekvő sorozatba. Ezt végezhetjük lineáris kereséssel, amikor $O(n)$ lépést végzünk. (Bináris keresésnél az összehasonlítások száma logaritmikus, de a mozgatásokra akkor is elmehet lineárisan sok lépés.)

   Megjegyzés: Egyetlen beszúrással is megoldható, hogyan?

8. Az $A[1:n]$ tömbben számokat tárolunk. Határozza meg $O(n \log n)$ lépésben

   (a) azokat az értékeket, amelyek egynél többször fordulnak elő;

   (b) a leggyakoribb értékeket (vagyis azokat, amelyeknél többször semelyik másik szám sem fordul elő a tömbben)!

Megoldás: (a) Ha a tömb rendezve van, akkor egyszerűbb a feladat, hiszen akkor az egyforma értékek egymás mellett vannak. Ezért először rendezzük a tömböt, például az összefésüléses rendezéssel, aminek lépésszáma $O(n \log n)$. Majd végigmegyünk a tömbön, és ha ugyanaz az érték több egymás utáni elemnél is előfordul, kiírjuk. Az algoritmusnak ez a része $O(n)$ lépés, tehát összesen $O(n \log n)$ lépést végzünk.

(b) Rendezzük a tömböt, mint előbb. Most azt kell meghatározni, hogy a rendezett tömbben melyik elem alkotja a leghosszabb blokkot. Ez egy számlálóval egyszerűen megoldható: nyilvántartjuk, hogy mennyi az eddigi legnagyobb blokk hossza, ott milyen érték volt, és számoljuk az aktuális blokk hosszát.

A rendezésen kívül itt is lineárisan sok lépés történik, ezért a lépésszám $O(n \log n)$.

9. Legyen adott egy egészekből álló $A[1 : n]$ tömb valamint egy b egész szám. Egy olyan $i, j \in \{1, \ldots, n\}$ indexpárt keresünk, melyre $A[i] + A[j] = b$. Hogyan lehet ezt $O(n \log n)$ időben megoldani?

Megoldás: Vegyük észre, hogy ha minden lehetséges indexpárt ki akarunk próbálni, akkor $\binom{n}{2} \in \Theta(n^2)$ lehetőség van, ami több, mint a kívánt lépésszám.

Itt is sokat segít a rendezettség, ezért rendezzük a tömböt $O(n \log n)$ lépésben, pl. összefésüléssel.

Ez után az $i = 1, 2, \ldots, n$ indexekre keressük a tömbben a $b - A[i]$ értéket. Ha ezt bináris kereséssel tesszük, akkor mindegyik keresés $\Theta(\log n)$ lépésszámú, tehát a lépésszám összesen $O(n \log n)$.

Megjegyzés: a rendezés utáni rész lineárisan is megoldható. Kiindulunk az $i = 1$, $j = n$ indexekből és általában, ha $A[i] + A[j] < b$, akkor az i értékét növeljük eggyel, ha pedig $A[i] + A[j] > b$ akkor a j értékét csökkentjük eggyel. Akkor hagyjuk abba, ha $A[i] + A[j] = b$ vagy ha $i > j$ lenne – ez utóbbi esetben nincs megoldás. (Miért?)

10. Az n méretű (nem feltétlenül rendezett) A tömb elemei különböző pozitív 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 $O(n \log n)$.

Megoldás: Vegyük észre, hogy egy rögzített k esetén, ha van megoldás, akkor a tömb legkisebb k eleme is megoldás. Ha ezek nem jók, akkor pedig nincs megoldás. Tehát azt kell vizsgálni, hogy van-e olyan k, hogy a legkisebb k elem összege legfeljebb k^3.

Ehhez rendezzük a tömböt $O(n \log n)$ lépésben, például összefésüléses rendezéssel. Legyen $k = 1$ és $T = A[1]$. Ha $T \leq k^3$, akkor készen vagyunk, különben növeljük k-t és adjuk T-hez az $A[k]$ számot. Ha egyik $k \leq n$ sem jó, akkor nincs megoldás.

A rendezés utáni rész n összehasonlítást és $O(n)$ összeadást használ, ezért összesen a lépésszám $O(n \log n)$.

11. Adott a síkon n pont, melyek koordinátái (a_1, b_1), (a_2, b_2), $\ldots$, (a_n, b_n). Olyan $P = (x, y)$ pontot keresünk a síkon, amire az $\sum_{i=1}^{n}(|a_i - x| + |b_i - y|)$ összeg minimális. Adjon algoritmust, amely $O(n \log n)$ lépésben meghatároz egy ilyen P pontot!

Megoldás: A felírt összeget minimalizálhatjuk úgy, hogy külön meghatározzuk az x koordinátában és külön az y koordinátában a minimumot. Tehát például olyan x kell, amire a $\sum_{i=1}^{n} |a_i - x|$ összeg minimális. Ehhez képzeljük el az a_i pontokat a számegyenesen. Ha x kisebb, mint a legkisebb a_i, akkor x-et jobbra mozgatva az összeg csökken. Amíg kevesebb a_i van előtte, mint mögötte, addig jobbra mozgatva x-et az összeg tovább csökken. Akkortól nő, amikor már több a_i van előtte, mint mögötte. Tehát az optimális x a rendezés szerinti középső a_i, ha n páratlan. Páros esetben mindegy, hogy hol van a középső intervallumon belül, például az $x = a_{n/2}$ választás jó.

Ezek után az algoritmus: rendezzük az a_i értékeket, és legyen x a rendezés szerinti $\lceil n/2 \rceil$ szám. Ugyanezt megcsináljuk y-ra: rendezzük a b_i értékeket, és legyen y a rendezés szerinti $\lceil n/2 \rceil$ szám.

Az algoritmus lépésszáma, pl. összefésüléses rendezés esetén $O(n \log n)$.

12. Adott a számegyenesen n intervallum, $[a_1, b_1], \ldots, [a_n, b_n]$. Azt akarjuk tudni, hogy együtt milyen hosszú részt fednek le a számegyenesből (azaz, hogy mennyi $\cup_{i=1}^{n}[a_i, b_i]$ összhossza). Adjon $O(n \log n)$ lépéses algoritmust ennek a hossznak a meghatározására!

Megoldás: Egy intervallum hossza $b_i - a_i$, de az unió lehet ezek összegénél kevesebb, ha az intervallumok

metszik egymást. Ezeket az átfedéseket kell valahogy kezelni, pontosabban megtalálni azokat a diszjunkt intervallumokat, amik az uniót alkotják.

Rendezzük a végpontokat (mind a $2n$ számot) úgy hogy tároljuk, melyik volt egy intervallum kezdőpontja és melyik egy intervallum végpontja. Az unió első része a legkisebb értéknél kezdődik (és ez szükségszerúen egy intervallum a_j kezdete). Menjünk a rendezett sorozatban addig, amíg először megegyezik a kezdő és végpontok száma. Ez egy végpontnál kövezkezik be (b_k), és vegyük észre, hogy itt ér véget az unió első része, ennek a résznek a hossza $b_k - a_j$. Ha még maradtak a sorozatban elemek, ugyanígy folytathatjuk.

A lépésszámhoz azt kell látni, hogy a rendezés után már csak végig kell menni a sorozaton, számlálva a kezdő- és végpontokat. A hossz meghatározásához legfeljebb n kivonás kell, majd ezek eredményét kell összeadni. A rendezés lépésszáma $O(2n \log(2n)) = O(n \log n)$, az utána következőké $O(n)$, tehát ez összesen $O(n \log n)$.

13. Adjon minél kevesebb összehasonlítást használó algoritmust, ami n elem közül megtalálja a legkisebbet és a legnagyobbat is!

Megoldás: Külön a legkisebbet és a legnagyobbat is meg lehet találni $n-1$ összehasonlítással, azaz megoldható $2n-2$ összehasonlítással. (Sőt $(2n-3)$-mal is. Miért?)

Ennél jobb a következő: előbb hasonlítsuk össze az elsőt a másodikkal, a harmadikat a negyedikkel, stb. Ez után vegyük minden párból a kisebb elemet. A legkisebb közülük kerül ki. Használjuk ezekre a minimumot kereső algoritmust. Hasonlóan a páronkénti nagyobb elemek között találjuk meg a maximumot. Ha páratlan sok elem van, akkor az utolsó pár helyett 3 elemből határozzuk meg a kicsit és nagyot.

A lépésszám: $\lfloor n/2 \rfloor + 2 + 2(\lfloor n/2 \rfloor - 1)$, ami kb. $1{,}5n$. Pontosabban, ha n páros, akkor $1{,}5n - 2$, páratlan esetben pedig $(n-1)/2 + 2 + 2(n-1)/2 - 2 = 1{,}5(n-1)$.

Megjegyzés: meggondolható, hogy $1{,}5n - 2$ kell is. Legyen B azoknak az elemeknek a száma, amelyek még lehetnek legkisebbek és legnagyobbak is, I, ami már csak legkisebb lehet (volt már nála nagyobb), A, ami már csak legnagyobb lehet (volt nála kisebb, E pedig a többi, aminél már volt kisebb és nagyobb is. Kezdetben $|B| = n$, a többi üres, a végén B üres, $|I| = |A| = 1$.

Két B-beli összehasonlításakor B kettővel csökken, I és A eggyel-eggyel nő. Ha egy B-belit nem B-belivel hasonlítunk, akkor B csökken és mindig lehet olyan eredmény, amikor I és A egyike nő, a másik nem csökken. Ha két I-belit vagy két A-belit hasonlítunk, akkor egyikük átkerül az E-be. A többi esetben mindig lehet olyan válasz, hogy sem I sem A nem változik. Ahhoz, hogy a B kiürüljön ezek szerint kell legalább $n/2$ összehasonlítás. A B-ből minden elem előbb az I-be vagy A-ba kerül, ahonnan viszont egyszerre csak egy tud kikerülni, tehát ezek (majdnem) kiürítéséhez kell összesen legalább $n-2$ összehasonlítás. Tehát együtt van legalább $1{,}5n - 2$.

14. Adjon minél kevesebb összehasonlítást használó algoritmust, ami n elem közül megtalálja a két legkisebbet!

Megoldás: Előbb a legkisebb, majd a többiből megint a legkisebb meghatározása $n - 1 + n - 2 = 2n - 3$ összehasonlítás. Ennél jobbat is kaphatunk. Gondoljunk egy kieséses bajnokságra. A győztes legyen a legkisebb. A második nyilván csak olyan lehet, akit a későbbi győztes vert ki (a többieknél van legalább kettő nagyobb). Tehát a másodikra körönként egyetlen jelölt van. A bajnokság lebonyolítható $\log n$ körben, ennyi közül kell kiválasztani a másodikat, tehát a feladat összesen $n + \log n - 2$ lépésben megoldható.