Algel II. gyakorlat

Dinamikusak vagyunk

2013. február 22.

5. **[Vizsga: 2007. május 29.] Legyen $w = w_1 w_2 \cdots w_n$ egy n betűből álló szó. Hívjuk részszónak w egy tetszőleges $w_i w_{i+1} \cdots w_{i+k}$ darabját ($1 \leq i \leq n-1$, $1 \leq k \leq n-i$). Adjunk algoritmust, ami $O(n)$ lépésben meghatározza az összes a-val kezdődő és b-re végződő részszó számát.**
Észrevesszük, hogy egy tetszőleges b karakter előtti összes a karakter meghatároz egy-egy ilyen szót. A szövegben haladva ha a-t találunk, akkor az eddigi a-k számát növeljük eggyel, ha b-t, akkor pedig a végeredményhez hozzáadjuk az a-számláló aktuális értékét. Az algoritmus egy végigolvasásból megvan, azaz $O(n)$, valamint helyes (ezt nagyon precízek indukcióval is bebizonyíthatják).

6. **Az $1, 2, \ldots, n$ számoknak adott két permutációja, $x_1, \ldots, x_n$ és $y_1, \ldots, y_n$. A két sorozat egy közös részsorozata egy $1 \leq i_1 < \cdots < i_k \leq n$, és egy $1 \leq j_1 < \ldots < j_k \leq n$ indexsorozattal adható meg, ahol $x_{i_m} = y_{j_m}$ teljesül minden $1 \leq m \leq k$ esetén. Adjunk $O(n^2)$ lépésszámú algoritmust, ami az x és y sorozatokban meghatároz egy leghosszabb közös részsorozatot!**
Felveszünk egy $n \times n$ méretű A mátrixot, ahol $A[i,j]$ jelentése a következő: az $x_1 x_2 \ldots x_i$-n és $y_1 y_2 \ldots y_j$ leghosszabb közös részsorozatának hossza. Kitöltés:

$$A[i,j] = \begin{cases} \max(A[i-1,j], A[i,j-1]), \text{ ha } x_i \neq y_j \\ A[i-1,j-1]+1, \text{ ha } x_i = y_j \end{cases}$$

A spec esetek (szélek) értelemszerűen. Ekkor a táblázatban a max. szám megadja a legnagyobb közös részsorozat hosszát, onnan átlósan visszalépkedve megkapjuk magát a sorozatot. Táblázat kitöltése $O(n^2)$, a megoldás megkeresése is megvan ennyiből (persze ezt nem muszáj utólag, nyilván a kitöltés közben is nyilvántarthatjuk).

7. **Egy játékban egy $n \times m$ rács bal felső sarkából kell eljutnunk a jobb alsó sarokba. Egy lépés során a rács mentén vizszintesen vagy függőlegesen tudunk a következő rácspontba lépni. Azonban van néhány kereszteződés, ahova nem szabad lépnünk. Ezek helyét az R tömb írja le, $R[i,j] = 1$, ha az (i,j) kereszteződésbe nem léphetünk, egyébként $R[i,j] = 0$. Adjunk $O(nm)$ futási idejű algoritmust annak meghatározására, hogy pontosan $n+m-2$ lépést téve a rácson hányféleképpen tudjuk a célt elérni!**
Mivel a két sarok közötti távolság $n+m-2$, ezért ténylegesen csak jobbra vagy lefele léphetünk. Szokásosan egy $n \times m$ méretű A mátrixot töltünk, ahol $A[i,j]$ jelentése: az i-edik sor j-edik oszlopára hányféleképpen tudunk szabályosan eljutni. Kitöltése:

$$\begin{aligned} A[1,1] &= 1 \\ A[i,j] &= \begin{cases} 0, \text{ ha } (i,j) \text{ tiltott} \\ A[i-1,j] + A[i,j-1], \text{ ha } (i,j) \text{ nem tiltott} \end{cases} \end{aligned}$$

Ez azért jó így, mert tiltott mezőre 0-féleképpen mehetünk, egy nem tiltott mezőre pedig csak fentről vagy balról. A megoldás $A[n,m]$-ben lesz található. Helyesség indukcióval könnyen belátható, lépésszám $O(nm)$, mert egy $n \times m$ méretű mátrixot töltünk ki egy menetben.

8. **Legyenek $a_1, a_2, \ldots, a_n$ tetszőleges egész számok és $m < n^2$ egész. Adjunk algoritmust, amely a bináris alakjukkal megadott $a_1, a_2, \ldots, a_n$ és m számokról eldönti polinom időben, hogy az $a_1, a_2, \ldots, a_n$ számok közül kiválasztható-e néhány úgy, hogy az összegük m-mel osztva egyet adjon maradékul!**

Vegyük fel az m-hez tartozó teljes maradékrendszert egy (mondjuk bool) A tömbbe! Kezdetben minden hely $false$. Az a_i szám kézbevételekor $\forall j : A[j] = true$ helyre állítsuk $A[(j + a_i) \pmod m]$-et $true$-ra, majd $A[a_i \pmod m]$-et is. Ha $A[1]$ $true$ lesz, akkor kiválasztható, ha a végére sem, akkor nem. Ezt be is kell bizonyítani! Azt állítom, hogy a_i kézbevétele után pontosan azok az $A[j]$-k vannak $true$-ra állítva, amik valahogy előállnak $a_1 \dots a_i$ közül néhány összegeként (mod m). Indukcióval nyilván igaz, hiszen a_1 választásával csak $a_1 \pmod m$ áll elő. Tfh k-ra igaz, ekkor nézzük $k + 1$-re. $a_1 \dots a_{k+1}$ közül néhányat választva vagy választjuk a_{k+1}-et, vagy nem. Ha nem, akkor pontosan azok állnak elő, amik $a_1 \dots a_k$ közül, ha igen, akkor egyrészt előállhat maga a_{k+1}, másrészt az eddig előálló számok mindegyikéhez hozzávehetjük a_{k+1}-et is. Ezért a feltétel igaz maradt, hiszen az összes lehetséges esetet lefedtük. A lépésszám polinomiális, hiszen $O(nm)$ műveletet végzünk (minden számhoz a teljes táblán végigmegyünk), ami ebben a spec. esetben azért polinomiális, mert $m = O(n^2)$. (Ha m-re nem lenne korlát, akkor ez akár exponenciális is lehetne.)

9. **Adjunk algoritmust, ami egy n csúcsú fában lineáris időben meghatározza a fában levő leghosszabb út hosszát!**
Szokásosan tetszőlegesen választhatunk egy gyökeret, így beszélhetünk szintekről és egyebekről. Minden csúcshoz egy számot fogunk rendelni (modjuk $l(i)$), lentről felfele haladva. Ez a szám azt jelenti, hogy a levelekből melyik a leghosszabb út az adott csúcsba. Egy v csúcs vizsgálatakor meg kell vizsgálni a rajta áthaladó utak közül a leghosszabbat, ez a két legnagyobb számmal rendelkező gyerekén (mondjuk i és j) keresztül megy, a hossza pedig $l(i) + l(j) + 1$. Ha ez hosszabb, mint az eddigi leghosszabb, akkor megjegyezzük, $l(v)$ pedig (ha $l(i)$ volt a maximális) $= l(i) + 1$. Bizonyítás indukcióval; lépészám: minden csúcsot csak akkor veszünk kézbe, amikor őt vizsgáljuk, és a szülőjének vizsgálatakor a két max. keresésben, így $O(3n) = O(n)$, ebbe a kezdeti gyökeresítés és a bejárás szervezése is belefér.

10. **[Vizsga: 2009. június 11.] A véges hosszú 0-1 sorozatok egy részét valamilyen szempontból jónak tekintjük, a többit rossznak. Van egy A algoritmusunk, mely adott n hosszú 0-1 sorozatról $O(\log(n!))$ lépésben megmondja, hogy a sorozat jó vagy rossz.**

    Adjon olyan eljárást, mely A-t felhasználva, ha adott egy m hosszú 0-1 sorozat, $y = y_1 y_2 \cdots y_m$, akkor eldönti, hogy y előáll-e néhány jó sorozat egymás után fűzéséből. Az eljárás lépésszáma összesen legyen $O(m^4)$.

    Segítség: $\log(n!) \approx O(n \log n) = O(n^2)$.
    Felveszünk egy T tömböt n mérettel. $T[i] = 1$ azt jelenti, hogy $y_1 \cdots y_i$ előáll jó sorozatokból. $T[i]$ kitöltése: $T[i] = A(1, i) \vee (T[1] \wedge A(2, i)) \vee \cdots \vee (T[j] \wedge A(j + 1, i)) \vee \cdots \vee (T[i - 1] \wedge A(i, i))$. Magyarul megnézzük, hogy az eddigi helyes részekhez hozzá tudjuk-e illeszteni az újat. A végeredmény $T[m]$-ben található. Ez $O(n^2)$ db meghívása A-nak legfeljebb n méretű inputtal, innen kijön a jó lépésszám.

11. **Egy n szóból álló szöveget kell sorokra tördelni. A szöveg i-edik szava ℓ_i karakterből áll, egy sor s karakter hosszú. Ha egy sor a szöveg i-edik szavával kezdődik és a j-edik szóval végződik, akkor az elválasztó szóközöket is figyelembe véve $t = s - (\ell_i + \ell_{i+1} + \cdots + \ell_j + j - i)$ üres hely marad a sor végén. Egy ilyen sor hibája legyen t^2. A tördelés hibája a nem üres sorok hibáinak összege. Adjunk $O(n^2)$ lépéses algoritmust egy legkisebb hibájú tördelés meghatározására! (A szavak sorrendje rögzített.)**
    Elég csúnyán hangzik, de nem az. Szokásosan valami olyasmire kell gondolni, hogy egy sor vizsgálatánál már rendelkezésünkre álljon az összes megelőző érdekes adat. Vegyünk fel egy D tömböt, ahol $D[i]$ értéke legyen az $1 \dots i$ szavak legkisebb hibájú tördelésének értéke! $D[0] = 0$ jelentse azt, hogy 0 szót 0 sorba hiba nélkül be tudunk rakni. Amikor épp az i-edik szót vizsgáljuk, akkor a következő lehetőségeink vannak: i-t külön sorba vesszük, az $i-1$-ig bezárólag pedig a lehető legjobb tördelést adjuk (jé, ez pont $D[i-1]$!); $i-1$-et és i-t vesszük egy sorba, az előttük levőket pedig a legjobban tördeljük (csak nem $D[i-2]$?);...; 1-től i-ig csinálunk egy sort.

Meg is van a képlet: $D[i] = \min_{j=1\ldots i}(t_{j,i}^2 + D[j-1])$, a helyességet a fenti gondolatmenet adja. $D[n]$ fogja az optimális tördelés értékét megadni. Ha meg is akarjuk kapni a tényleges tördelést, akkor egy T tömbben nyilvántarthatjuk, hogy adott i-hez $D[i]$-nél hogyan kaptuk a minimumot, azaz hol van a sortörés. $T[n]$-től visszafele lépkedve megkapjuk a sortörések helyét. Mivel az i-edik szónál 1-től i-ig végignézzük az eddigi tömböt (azaz $1, 2, \ldots, n$ műveletet végzünk), a lépésszám $O(n^2)$ lesz.

12. **Adottak $M_1, M_2, \ldots, M_n$ munkák $H_1, H_2, \ldots, H_3$ határidőkkel és $P_1, P_2, \ldots, P_n$ profitokkal. Mindegyik munka elvégzése pontosan 1 napunkba kerül (egy napon csak egy munkát végezhetünk el). Adjunk $O(n^2)$ lépésszámú algoritmust, amely meghatározza, hogy mely munkákat vállaljuk el a profit maximalizálása érdekében!**
Alakítsuk a határidőket a mai naptól induló számozásra (ha szükséges). Legyen $A[i]$ a legfeljebb i indexű feladatokat használó maximális profitú ütemezés: $A[0] = 0$. Ha $i > 0$, akkor $A[i]$ úgy kapható, hogy az i indexű feladatot megkíséreljük a határidejétől visszafelé betenni a naptárunkba. Ha van üres hely, akkor odatesszük és $A[i] = A[i-1] + P_i$. Ha nincs üres hely, de a legkisebb profitú munka profitja (P_j) kisebb, mint P_i, akkor az M_j munka helyére tesszük be a M_i munkát. Ez $O(n^2)$.