Ládapakolás, dinamikus programozás

1. A LÁDAPAKOLÁS feladatban legyenek a súlyok: $s_1 = 0{,}4$; $s_2 = 0{,}7$; $s_3 = 0{,}1$; $s_4 = 0{,}6$. Hajtsa végre erre (a) az FF algoritmust; (b) az FFD algoritmust! (c) Hány ládát használ az optimális pakolás?

   Megoldás: (a) 3 láda lesz: $0{,}4 + 0{,}1$, $0{,}7$, $0{,}6$.
   (b) 2 láda lesz: $0{,}7 + 0{,}1$, $0{,}6 + 0{,}4$.
   (c) Nyilván az utóbbi optimális, hiszen a súlyok összege $0{,}4 + 0{,}7 + 0{,}1 + 0{,}6 = 1{,}8 > 1$, azaz 1 láda nem elég.

2. A LÁDAPAKOLÁS feladatban legyen két súly 0,34 és négy súly 0,33 értékű. Hajtsa végre erre (a) az FFD algoritmust! (b) Hány ládát használ az optimális pakolás?

   Megoldás: (a) 3 láda lesz: $0{,}34 + 0{,}34$, $0{,}33 + 0{,}33 + 0{,}33$, $0{,}33$.
   (b) 2 láda elég, mindkettő $0{,}34 + 0{,}33 + 0{,}33$. Mivel ezek a ládák tele vannak, ez nyilván optimális.

3. Tekintsük a LÁDAPAKOLÁS problémának azt a speciális esetét, amikor minden súly 1/2 vagy 1. Igazolja, hogy ez a változat P-ben van!

   Megoldás: Az 1 méretű tárgyak 1-1 ládába kerülnek, az 1/2-eseket meg tetszőlegesen párosíthatjuk. A ládák száma: $n_1 + \lceil (n - n_1)/2 \rceil = \lceil (n + n_1)/2 \rceil$, ha n az összes, n_1 az 1 méretű tárgyak száma.

4. Egy f fokú létrán bizonyos fokok annyira rozogák, hogy ha rálépünk, leszakadnak. Szerencsére tudjuk, hogy melyik fokok ilyenek, hova nem szabad lépnünk. Egy lépéssel legfeljebb 3 fokot tudunk lépni. Adjon dinamikus programozást használó algoritmust ami meghatározza, hogy (a) a létra aljától fel tudunk-e jutni a létra legfelső fokára! (b) a létra aljától hányféleképpen tudunk feljutni a létra legfelső fokára!
   (Feltehető, hogy a legfelső fokra rá szabad lépni.) Mennyi az algoritmusok lépésszáma?

   Megoldás: Legyen NR az a tömb amiben $NR[i]$ akkor igaz, ha az i-edik fok nem rozoga.

   (a) A részfeladatok azok lesznek, hogy az i-edik fokra fel tudunk-e jutni ($1 \leq i \leq f$). Az $F[i]$ táblázatot töltjük ki úgy, hogy $F[i]$ igaz, ha fel tudunk az i-edik fokra jutni.

   Vegyük észre, hogy a feladat az $F[f]$ értéket kérdezi.

   A kitöltés kezdete: $F[0] =$ igaz, azaz a földről nem esünk le, $F[1] = NR[1]$.

   A rekurzió: mivel az i-edik fokra eggyel, kettővel vagy hárommal lejebbről jöhetünk, pontosan akkor tudunk az i-edikre lépni, ha a három megelőző valamelyikére el lehet jutni és az i-edik fok nem rozoga. Ezért az általános rekurzió: $F[i] = NR[i] \wedge (F[i-1] \vee F[i-2] \vee F[i-3])$.

   Látszik, hogy ez csak $i \geq 3$ esetben működik. Az $i = 2$ esetre hasonló formula jó, csak ott nem szerepel a nem létező $F[i-3]$, azaz $F[2] = NR[2] \wedge (F[1] \vee F[0])$.

   Ha a táblázatot $i = 0, 1, 2, \ldots, f$ sorrendben töltjük ki, akkor minden elemnél konstans sok lépést használunk, az algoritmus lépésszáma $O(n)$.

   (b) Hasonló az (a) részhez, de most nem logikai változó kell. Legyen $M[i]$ az, hogy hányféleképpen tudunk feljutni az i-edik fokra.

   A feladat megoldását $M[f]$ szolgáltatja.

   A kitöltés kezdete: legyen minden $M[i] = 0$ ha $i > 0$ és $M[0] = 1$. Mivel az i-re lépés csak az előző három valamelyikéről történhet, ezért a lehetséges feljutások számát a három előző szám összegeként kapjuk meg. Ebben úgy vesszük figyelembe a rozoga fokokat,hogy azoknál az M értéke 0 lesz, tehát nem adnak semmit a keresett számhoz. Ezek alapján:

   Ha $NR[1]$ igaz, akkor legyen $M[1] = 1$, ha $NR[2]$ igaz, akkor $M[2] = M[1] + M[0]$ és általában, ha $NR[i]$ igaz, akkor legyen $M[i] = M[i-1] + M[i-2] + M[i-3]$.

   Most is $i = 0, 1, 2, \ldots, f$ sorrendben kell kitölteni a táblázatot és akkor a használni kívánt értékek már rendelkezésünkre állnak, minden i-re konstans sok lépés lesz, azaz összesen $O(n)$.

5. Egy $n \times n$ méretű táblázat mezőin lépkedünk a bal alsó sarokból a jobb felső sarokba úgy, hogy egy lépésben a táblázatban vagy felfelé vagy jobbra egyet lépünk, de van néhány ,,tiltott" mező, ahova nem léphetünk. Adjon egy dinamikus programozást használó eljárást, ami meghatározza, hogy hányféleképpen érhetünk célba!

   Megoldás: Definiáljunk egy $n+1$ sorból és oszlopból álló T táblázatot, amelyben a $T[i,j]$ elem jelentése, hogy hányféleképpen tudunk eljutni az (i,j) mezőbe.

   Legyenek a T 0. sorában és oszlopában az értékek nullák, $T[1,1] = 1$.

   A rekurziós formulához vegyük észre, hogy az (i,j) mezőbe csak az $(i-1,j)$ vagy az $(i,j-1)$ mezőkből léphetünk és ha T-ben a tiltott mezőknek megfelelő értékeket 0-ra állítjuk, akkor a

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

   formula $i \geq 1$, $j \geq 1$ esetén helyes lesz.

   Ez a táblázat kitölthető soronként, hiszen akkor az (i,j)-hez érve a két felhasználandó érték már rendelkezésünkre áll.

   Lépésszám: a táblázat mérete $(n+1)^2 \in O(n^2)$. Minden elemhez konstans sok művelet szükséges, ezért az összes lépésszám is $O(n^2)$.

6. Egy $n \times n$ méretű táblázat minden eleme egy pozitív egész szám. A táblázat bal alsó sarkából akarunk eljutni a jobb felső sarkába úgy, hogy egy lépésben a táblázatban vagy felfelé vagy jobbra egyet lépünk. Azt szeretnénk, hogy a lépegetés során látott elemek növekvő sorrendben kövessék egymást. Adjon $O(n^2)$ lépésszámú algoritmust, ami meghatározza, hogy (a) hány a szabályoknak megfelelő út van! (b) mekkora a legnagyobb értékű, a szabályoknak megfelelő út, ha egy út értéke a benne szereplő számok szorzata!

   Megoldás: (a) Dinamikus programozást használunk: Jelölje A az eredeti (adott) táblázatot. Készítünk egy $n \times n$ méretű T táblázatot, amiben a $T[i,j]$ értéke az lesz, hogy hányféleképpen tudunk a szabályoknak megfelelően A-ban az i-edik sor j-edik elemébe jutni. Ha $A[1,1]$ a bal alsó mező és $A[n,n]$ a jobb felső, ekkor $T[n,n]$ adja a keresett választ.

   Legyen $T[1,1] = 1$. Mivel az (i,j) pozícióba csak balról vagy lentről jöhetünk, akkor van jó út, ha ezek közül legalább az egyikbe el lehet jutni, és onnan az utolsó lépés is érvényes, azaz teljesül, hogy az $A[i,j]$ nagyobb a megelőző elemnél.

   Ezért az első sor $j > 1$ esetben $T[1,j] = T[1,j-1]$, ha $A[1,j] > A[1,j-1]$, különben meg $T[1,j] = 0$. (A sor egy darabig 1 lesz, egy idő után meg 0, ha volt egy nem növekvő lépés.) Hasonlóan. az $i > 1$ esetben $T[i,1] = T[i-1,1]$, ha $A[i,1] > A[i-1,1]$, különben meg $T[i,1] = 0$.

   Az általános rekurzió, amikor $i,j > 1$:

   - ha $A[i,j] > A[i-1,j]$ és $A[i,j] > A[i,j-1]$, akkor $T[i,j] = T[i-1,j] + T[i,j-1]$
   - ha $A[i,j] > A[i-1,j]$ és $A[i,j] \leq A[i,j-1]$, akkor $T[i,j] = T[i-1,j]$
   - ha $A[i,j] \leq A[i-1,j]$ és $A[i,j] > A[i,j-1]$, akkor $T[i,j] = T[i,j-1]$
   - ha $A[i,j] \leq A[i-1,j]$ és $A[i,j] \leq A[i,j-1]$, akkor $T[i,j] = 0$.

   Ha az első sor és oszlop meghatározása után a táblázatot soronként töltjük ki, akkor mindig rendelkezésünkre áll a szükséges információ, így minden elem kiszámolása konstans sok műveletet fog használni.

   Mivel a T tömb mérete n^2 és minden elemének kitöltése konstans sok művelet, ezért a lépésszám $O(n^2)$.

   (b) Az előzőhöz hasonló, de most a maximális értéket tároljuk, ezért minden érvényes lépésnél szorozni kell az aktuális A értékkel, az összeadás helyett pedig a nagyobb értéket kell választani.

   Ha ez most egy S tömb, akkor $S[1,1] = A[1,1]$, mivel eddig ez az érték. Az első sor és oszlop kitöltése így alakul: $S[1,j] = S[1,j-1] \cdot A[1,j]$, ha $A[1,j] > A[1,j-1]$, különben meg $S[1,j] = 0$. Hasonlóan az $i > 1$ esetben $S[i,1] = S[i-1,1] \cdot A[i,1]$, ha $A[i,1] > A[i-1,1]$, különben meg $S[i,1] = 0$.

   A továbbiakban $i,j > 1$, ekkor pedig

   - ha $A[i,j] > A[i-1,j]$ és $A[i,j] > A[i,j-1]$, akkor $S[i,j] = A[i,j] \cdot \max\{S[i-1,j], S[i,j-1]\}$

- ha $A[i,j] > A[i-1,j]$ és $A[i,j] \le A[i,j-1]$, akkor $S[i,j] = A[i,j] \cdot S[i-1,j]$
- ha $A[i,j] \le A[i-1,j]$ és $A[i,j] > A[i,j-1]$, akkor $S[i,j] = A[i,j] \cdot S[i,j-1]$
- ha $A[i,j] \le A[i-1,j]$ és $A[i,j] \le A[i,j-1]$, akkor $S[i,j] = 0$.

Ha az első sor és oszlop meghatározása után a táblázatot soronként töltjük ki, akkor mindig rendelkezésünkre áll a szükséges információ, így minden elem kiszámolása konstans sok műveletet fog használni. Ezért a teljes lépésszám most is $O(n^2)$.

7. Legyen $s_1 s_2 \dots s_n$ és $t_1 t_2 \dots t_m$ egy n és egy m hosszú karaktersorozat. Azt szeretnénk, hogy az $n \times m$ méretű A mátrix $A[i,j]$ eleme tartalmazza azt a legnagyobb k számot, melyre az $s_1 s_2 \dots s_i$ és a $t_1 t_2 \dots t_j$ sorozatok utolsó k karaktere megegyezik. Adjon eljárást, ami az A tömböt $O(nm)$ lépésben kitölti.

Megoldás: Ha $i = 1$ vagy $j = 1$, akkor egyszerű, hiszen az egyik sorozat 1 elemű, az érték csak 0 vagy 1 lehet.

$$A[1,j] = \begin{cases} 1, & \text{ha } s_1 = t_j \\ 0, & \text{ha } s_1 \neq t_j \end{cases} \qquad A[i,1] = \begin{cases} 1, & \text{ha } s_i = t_1 \\ 0, & \text{ha } s_i \neq t_1 \end{cases}$$

A továbbiakhoz azt kell észrevenni, hogy ha az s_i és t_j különbözik, akkor nyilván $k = 0$, egyébként pedig a leghosszabb ilyen közös rész az eggyel rövidebb kezdőszeletekre kapott egyező rész kiterjesztése,

$$A[i,j] = \begin{cases} 1 + A[i-1,j-1], & \text{ha } s_i = t_j \\ 0, & \text{ha } s_i \neq t_j \end{cases}$$

Amennyiben a táblázatot soronként töltjük ki, akkor mindig rendelkezésünkre áll a szükséges információ, egy elem kiszámolása konstans művelet, a táblázat mérete nm, ezért az egész összesen $O(nm)$.

8. Egy n és egy m karakterből álló szövegben meg akarjuk találni a legnagyobb azonos darabot, azaz ha az egyik szöveg $a_1 a_2 \cdots a_n$ és a másik $b_1 b_2 \cdots b_m$, akkor olyan $1 \le i \le n$ és $1 \le j \le m$ indexeket keresünk, hogy $a_{i+1} a_{i+2} \dots a_{i+t} = b_{j+1} b_{j+2} \dots b_{j+t}$ teljesüljön a lehető legnagyobb t számra. Adjon erre a feladatra $O(nm)$ lépést használó algoritmust.

Megoldás: Vegyük észre, hogy ha az előző feladatban leírt táblázatot elkészítjük erre az esetre, akkor a táblázatbeli legnagyobb érték pont a megfelelő t lesz. Ennek helye megadja a megfelelő $i+t$ és $j+t$ indexeket, amikből i és j is megkapható.

A táblázat kitöltése az előző feladat szerint $O(nm)$ lépés. A maximális érték megtalálása további $nm - 1$ összehasonlítással megoldható, ezek után i és j kiszámolás konstans lépés. Ez összesen is $O(nm)$.

9. Adott egy n és egy m hosszú 0-1 sorozat, $a_1, a_2, \dots, a_n$, illetve $b_1, b_2, \dots, b_m$. Ezek alapján egy T tömböt töltöttünk ki a következő módon:

Ha $0 \le i \le n$, akkor $T[i,0] = 0$. Ha $0 \le j \le m$, akkor $T[0,j] = 0$.

Ha $1 \le i \le n$ és $1 \le j \le m$, akkor $T[i,j] = \begin{cases} T[i-1,j-1] + 1 & \text{ha } a_i = b_j \\ \max\{T[i,j-1], T[i-1,j]\} & \text{ha } a_i \neq b_j \end{cases}$

Mi a jelentése a $T[i,j]$ értéknek! A két sorozatnak milyen tulajdonságát adja meg a $T[n,m]$ érték?

Megoldás: $T[i,j]$ = az $a_1 \cdots a_i$ és $b_1 \cdots b_j$ sorozatokban mekkora a legnagyobb közös részsorozat hossza, ezért $T[n,m]$ a két teljes sorozatban a leghosszabb közös részsorozat hossza.

Ez igaz, ha $i = 0$ vagy $j = 0$. A rekurziós képlet meg azt mutatja, hogy a legnagyobb közös részsorozatban vagy $a_i = b_j$, és a korábbiakból kell a leghosszabb, ezért a hossz $T[i-1,j-1] + 1$. Ha meg $a_i \neq b_j$, akkor vagy a b_j vagy az a_i nincs benne a leghosszabb közös részsorozatban, ezt mutatja a második lehetőség.

Megjegyzés: ilyen feladatoknál érdemes valami egyszerű példán megnézni mi történik, az alapján megtippelni a szabályt (amit persze be is kell bizonyítani).

10. Éllistával adott egy n pontú e élű G irányított gráf, ami egy DAG. Adjon $O(n+e)$ lépésszámú algoritmust, ami minden v pontra meghatározza azoknak az utaknak a számát

    (a) amelyek egy rögzített s pontból v-be visznek!

    (b) amelyek v-ből egy rögzített t pontba visznek!

Megoldás: A gráfunk egy DAG (azaz nincs benne irányított kör), tehát a csúcsoknak van topologikus sorrendje, ami mélységi bejárás segítségével $O(n+e)$ lépésben meghatározható. Legyen egy ilyen sorrend $v_1, v_2, \ldots, v_n$. A továbbiakban dinamikus programozást használunk: sorban az $i = 1, 2, \ldots, n$ indexekre meghatározzuk a v_i-be vezető utak számát, ezt fogja az $U[i]$ jelölni. Ebben használjuk, hogy egy tetszőleges v_i-be csak kisebb indexű csúcsból vezet él.

Legyen az s indexe m. Ekkor minden $i < m$ esetben világos módon $U[i] = 0$, továbbá $U[m] = 1$. Kezdetben a többi $U[i]$ érték is legyen 0. Egy $v_i \neq s$ csúcsba érő útnak ($i > m$) van utolsó előtti csúcsa. A v_i-be érő utak számát úgy kapjuk, ha összeadjuk hányféleképpen tudtunk eljutni a lehetséges utolsó előtti csúcsokba. Ennek megfelelően a rekurzió $i = m+1, \ldots, n$: $U[i] = \sum_{j:(v_j,v_i)\in E} U[j]$.

Vegyük észre, hogy így valóban csak az s-ből induló utakat számoljuk, minden más esetben 0 marad $U[i]$ értéke.

Az $U[i]$ kiszámolása a v_i-be befutó élek számával arányos. Az összes csúcsra összeadva kapjuk, hogy ez e-vel arányos. Mivel lehetnek bemenő él nélküli csúcsok is, amiket szintén kezelnünk kell, a teljes lépésszám $O(n+e)$.

(b) 1. megoldás: ez ugyanaz, mint az (a) rész, csak „fordítva". Készítsük el a fordított gráfot, amikor minden élet megfordítunk. Az így kapott gráfra alkalmazzuk az (a) rész megoldását.

2. megoldás: a gráf megfordítása nélkül az U tömböt kitölthetjük visszafelé: ha $t = v_T$, akkor $U[i] = 0$, amennyiben $i > T$, $U[T] = 1$, és az $i = T-1, \ldots, 1$ esetben $U[i] = \sum_{j:(v_i,v_j)\in E} U[j]$

1. Megjegyzés: Vegyük észre, hogy ez egy lineáris algoritmus, hiszen $O(n+e)$ az éllista teljes mérete.

2. Megjegyzés: Igazából éllista alatt többnyire a kimenő élekből képzett listákat értjük, itt meg a bemenő élek listái kellenek. Ezt az előzőből egyszerűen $O(n+e)$ lépésben előállíthatjuk. A (b) esethez nem is kell megfordítani a gráfot, az eredeti bemenő éllista, ha kimenőnek tekintjük épp a fordított gráfot adja.