

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 feladat azon speciális esetét, amikor minden tárgyra $1 \geq s_i > 1/3$ teljesül. Adjon polinom idejű algoritmust, ami megadja a pakoláshoz szükséges 1 méretű ládák számának minimumát!

Megoldás:

I. ágyúval verébre

Az $s \geq 1/3$ súlyú tárgyakból legfeljebb kettő kerülhet egy ládába. Készítsünk el egy gráfot, melynek csúcsai a tárgyaknak felelnek meg. Közülük kettő akkor legyen összekötve, ha összegük ≤ 1 . A pakolások megfelelnek a gráfbeli párosításoknak. Akkor használunk minimális számú ládát, ha a megfelelő párosítás maximális. Maximális párosítást gráfokban (nem csak párosokban) lehet polinomidőben találni, így a feladat is megoldható polinom lépésben. (Ennél azért jobbat is lehet mondani, ha kihasználjuk a gráf sajátosságait. Az hogy hogyan, látszik majd a következő megoldásból.)

II. Gondolkozzunk

Rendezzük a tárgyakat súlyuk szerinti csökkenő sorrendbe, $s_1 \geq \dots \geq s_n$. Legyen $i = 1, j = n$. Ha $s_i + s_j \leq 1$, rakjuk ezeket egy ládába, i -t növeljük, j -t csökkentjük eggyel. Ha $s_i + s_j > 1$, az i . tárgy egyedül lesz egy ládában, növeljük i -t eggyel. Ha $i \geq j$, az eljárás véget ért, megadtunk egy pakolást (lineáris időben!). Most belátjuk, hogy ez optimális: elég megmutatni, hogy van olyan optimális pakolás, amelynél az s_1 -et tartalmazó ládának ugyanaz a tartalma, mint az általunk javasolt elhelyezésnél. Ebből – s_1 -et és esetleg s_n -et elhagyva – kézenfekvő indukcióval következik az állítás. Ha s_1 a mi módszerünkénél egyedül van a ládájában, akkor minden más bepakolásnál is egyedül kell lennie, ekkor tehát készen vagyunk. Ha egy pakolásnál s_1 társa s_j ($1 < j < n$), akkor világos, hogy s_j és s_n helyet cserélhet anélkül, hogy túllépnénk a korlátot. Legyen ugyanis s_n társa s . Ekkor $1 \geq s_1 + s_j \geq s_1 + s_n$ és $1 \geq s_1 + s_j \geq s + s_j$. Még egyszerűbben tudunk elbánni egy olyan pakolással, amelyben s_1 és/vagy s_n egyedül van.

III. vizsgáljuk meg a tanultakat

Az FFD ebben az esetben optimális eredményt ad. Hasonló gondolatmenettel, mint az előbb meg lehet mutatni, hogy az FFD-től legkevésbé eltérő optimális megoldás megegyezik vele.

4. Egy f fokú létrán bizonyos fokok annyira rozogák, hogy ha rálépünk, leszakadnak. Szerencsére tudjuk, hogy melyik fokok ilyenek, hova nem szabad lépni. 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: $F[0] = \text{igaz}$, azaz a földről nem esünk le, $F[1] = NR[1]$.

A rekurzió: mivel az i -edik fokra eggyel, kettővel vagy hárommal lejjebből 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 $i = 0, 1, 2, \dots, f$ 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 $i = 0, 1, 2, \dots, f$ sorrendben kell kitölteni a táblázatot és akkor a használni kívánt értékek már rendelkezésünkre állnak, minden i -re konstans sok **összeadás** lesz, azaz összesen $O(n)$ **összeadás**, az egyszerűség kedvéért ezt tekinjük a lépésszámnak is.

Ha azt is figyelembe vesszük, hogy az összeadandó számok nem feltétlenül konstans méretűek, akkor lépésszám nagyobb lesz. Indukcióval könnyen belátható azonban, hogy a legnagyobb szám is legfeljebb 3^n . Ennek leírásához $O(n)$ számjegy elegendő. Mivel az összeadást a jegyek számában lineáris algoritmussal el tudjuk végezni, az algoritmus teljes lépésszáma $O(n^2)$.

5. Egy $m \times m$ 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 $m + 1$ 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 $(m+1)^2 \in O(n)$, hiszen az input mérete $n = m^2$. Minden elemhez konstans sok összeadás szükséges, ezért az összeadások száma is $O(n)$. Itt is figyelniünk kell azonban a számok méretére. Ha egyetlen tiltott mező sincs, akkor a legnagyobb szám $\binom{2m}{m}$. Mivel $\sum_{i=1}^{2m} \binom{2m}{i} = 2^{2m}$, ezért $\binom{2m}{m} \leq 2^{2m}$. Ennek jegyeinek száma $O(m)$. Így a lépésszám $O(m^3)$, azaz $O(n^{1.5})$.

Megjegyzés: A továbbiakban az egyszerűség kedvéért feltesszük, hogy aritmetikai műveleteket 1 lépésben el tudunk végezni tetszőleges nagyságú számokkal.

6. Tekintsük az RH problémának azt a változatát, amikor adottak az $a_1, a_2, \dots, a_n$ 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 $I \subseteq \{1, \dots, n\}$, melyre $\sum_{i \in I} a_i = b$. Mutassa meg, hogy ez a változat P-ben van!

Megoldás: Vegyük észre, hogy ha $b > a_1 + a_2 + \dots + a_n$, akkor biztos nincs megoldás, tehát a $b \geq n^3$ esetben az algoritmus térjen vissza a „nem” válasszal.

Ha viszont $b < n^3$, akkor vegyünk fel egy $n + 1$ sorból és $b + 1$ oszlopból álló T táblázatot, melynek sorait 0-tól n -ig, oszlopait 0-tól b -ig indexeljük. Legyen $T[0, 0] = \text{igen}$ és $T[0, c] = \text{nem}$ minden $1 \leq c \leq b$ értékre.

A táblázat $T[i, c]$ értéke azt mutassa meg, hogy az $a_1, a_2, \dots, a_i$ számok közül néhány összegeként a c számot elő lehet-e állítani ($1 \leq i \leq n, 1 \leq c \leq b$). A feladat a $T[n, b]$ értékét kérdezi.

Vegyük észre, hogy a megadott nulladik sor is megfelel az értelmezésnek: a 0 hosszú sorozatból tetszőleges $c > 0$ szám csak nullaféleképpen 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 *igen* 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, vagy ha $c \geq a_i$, akkor az a_i számot is használhatjuk, ekkor a korábbiakból a maradék $c - a_i$ összeget kell nézni, hogy előállítható-e. Tehát: $T[i, c] = T[i - 1, c] \vee T[i - 1, c - a_i]$, 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 $i = 1, \dots, n$ értékre kitöltjük a $c = 0, 1, \dots, b$ mezőket.

A táblázat mérete $(n + 1) \cdot (b + 1) \in O(nb)$. Minden elemének kitöltése konstans lépés, tehát az összes idő $O(nb)$. A b -re alkalmazott korlát miatt $O(n^4)$ időben tudjuk kitölteni a táblázatot, tehát ez az eljárás polinomiális.

7. 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

- amelyek egy rögzített s pontból v -be visznek!
- 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 $v_1, v_2, \dots, v_n$. A továbbiakban dinamikus programozást használunk: sorban az $i = 1, 2, \dots, n$ indexekre meghatározzuk a v_i -be vezető utak számát, ezt fogja az $U[i]$ jelölni. (Az egyszerűség kedvéért itt sem vesszük figyelembe a lépésszámnál, hogy a számok mérete nem feltétlen lesz konstans méretű, mindegyiket 1 méretűnek tekintjük.) 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 $i = m + 1, \dots, n$: $U[i] = \sum_{j: (v_j, v_i) \in E} U[j]$.

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 kezelniük 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 $i = T - 1, \dots, n$ 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 kelljenek. 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.
8. Adott n pozitív egész szám, $a_1, a_2, \dots, a_n$. Az $n+1$ sorból és $b+1$ oszlopból álló T táblázat sorait 0-tól n -ig, oszlopaikat 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 $a_1, a_2, \dots, a_i$ 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 nullaféleképpen 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 $i = 1, \dots, n$ értékre kitöltjük a $c = 0, 1, \dots, b$ mezőket.

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

9. 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 lentől 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 $A[i, j] \leq A[i, j-1]$, akkor $S[i, j] = A[i, j] \cdot S[i-1, j]$
- ha $A[i, j] \leq A[i-1, j]$ és $A[i, j] > A[i, j-1]$, akkor $S[i, j] = A[i, j] \cdot S[i, j-1]$
- ha $A[i, j] \leq A[i-1, j]$ és $A[i, j] \leq A[i, j-1]$, 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 $O(n^2)$.

10. 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.

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

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 $O(nm)$.

11. Egy n és egy m karakterből álló szövegben meg akarjuk találni a legnagyobb azonos darabot, azaz ha az egyik szöveg $a_1 a_2 \dots a_n$ és a másik $b_1 b_2 \dots b_m$, akkor olyan $1 \leq i \leq n$ és $1 \leq j \leq m$ 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 a 10. 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 $O(nm)$.