Rónyai Lajos

Tablók – algoritmusok, növő sorozatok, genetikai sorozatok

című 2006. március 7.-én tartott előadása alapján írta

Erdélyi Márton és Hraskó András

Ajánló
Kapcsolódó anyag honlapunkon: Babai László 2004. májusi előadása
http://matek.fazekas.hu/portal/tanitasianyagok/Babai_Laszlo/Szamitas_0405/eloadas_0405.html

1. Rendezések

1. feladat

Van öt golyónk, amelyek különböző tömegűek, de ránézésre nem állapítható meg, hogy melyik könnyebb és melyik nehezebb. Ennek megállapításához egy kétkarú mérleg áll rendelkezésünkre. Rakjuk a golyókat tömegük szerinti növekvő sorrendbe úgy, hogy a kétkarú mérleget a lehető legkevesebbszer használjuk!

A megoldást lásd itt: http://matek.fazekas.hu/portal/eloadas/2005/5golyo.html

Fogalmazzuk meg a feladatot általánosan!

2. feladat (Rendezési feladat)

Legyenek $a_1, a_2, a_3, \ldots, a_n$ különböző egész számok! Rendezzük át őket minél gyorsabban, azaz minél kevesebb páronkénti összehasonlítással a $b_1 < b_2 < b_3 < \ldots < b_n$ sorozatba, ahol a *b* sorozat az *a* sorozat egy átrendezése (permutációja).

2/1. módszer (Buborék-rendezés)

Az első néhány összehasonlítással a legnagyobb elemet tesszük hátra, utána a maradékból a legnagyobbat, ezt ismételjük egészen addig, míg a legkisebbig jutunk . Mindig szomszédos elemeket hasonlítunk össze, előlről indulunk, hátrafelé megyünk.

Például: (**4**, **7**, 2, 5, 6)® (4, **7**, **2**, 5, 6)® (4, 2, **7**, **5**, 6)® (4, 2, 5, **7**, **6**)® (4, 2, 5, 6, 7)

Az első körben ez legfeljebb *n*-1 összehasonlítás, a másodikban *n*-2, az *i*.-ben *n*-*i*, tehát összesen $(n-1)+(n-2)+\ldots+1=\frac{n(n-1)}{2}$.

Van gyorsabb módszer is:

2/2. módszer (Beszúrásos rendezés)

Egyesével rakjuk be az elemeket úgy, hogy sorban legyenek: először a_1-et, utána megnézzük, hogy a_2 előtte, vagy utána van-e, a_3 már 3 helyen is lehet, utána a többit is hasonlóan.

Például: (4, 7, 2, 5, 6) ® (**4**) ® (4, **7**) ® (**2**, 4, 7) ® (2, 4, **5**, 7) ® (2, 4, 5, **6**, 7)

Tehát kell egy jó algoritmus egy új elem beszúrásához. Nyilván nem érdemes a már meglévő halmaz összes elemével összehasonlítani, a számbarkochbás módszer sokkal jobb. Mindig a(z egyik) középsővel hasonlítjuk össze.

Minden összehasonlításnál a halmaz méretét felezzük. Ha az eredeti halmaz *S*, az *i*. összehasonlítás után maradt pedig S_i, $|S_1| \le \frac{|S|}{2}, |S_{i+1}| \le \frac{|S_i|}{2}$, tehát egy *n* elemű halmazba beszúráskor legfeljebb $\lceil \log_2 n \rceil + 1$ összehasonlítás kell.

Például **S**={2, 4, 5, 7} -be beszúrni a 6-ot: 4<6, így már csak az $\mathbf{S_1}$={5, 7}-be kell, de 5<6, azaz csak azt kell eldönteni, hogy 7 előtt, vagy 7 után jön e ($\mathbf{S_2}$={7}). 4 elemű halmazba 3 összehasonlítás kellett.

Összesen ez legfeljebb

$$0+(\log_2 1+1)+(\log_2 2+1)+\ldots+(\log_2(n-1)+1)=(n-1)+\log_2(n-1)! \le n(\log_2 n+1)$$

összehasonlítás.

Ennek a módszernek programozói szemszögből nagy hátránya, hogy ha **S**-et tömbnek tekintjük minden elem beszúrásánál az utána levőket is át kell pakolni.

Még egy módszert megnézünk érdekességképpen. Ez a módszer eddig abszolút győztes, akkor is, ha a gyorsaságba a pakolgatás is beleszámít.

2/3. módszer (Összefésüléses rendezés)

Itt két rendezett halmazból csinálunk egy nagyot, például: (3, 6, 9, 11); (2, 4, 10) ® (2, 3, 4, 6, 9, 10, 11). Hasonlítsuk össze először a két legkisebb elemet! Kapjuk, hogy 2<3. Így biztosak lehetünk benne, hogy az egyesített listában a 2 lesz a legkisebb elem. Felejtsük is el! Foglalkozzunk a (3, 6, 9, 11); (4, 10) rendezett halmazokkal. Hasonlítsuk össze megint a két legkisebbet! 3<4, tehát keresett listánk (egyelőre) a (2,3) és most már csak a (6, 9, 11); (4, 10) halmazokat fésüljük...

Ha a két halmaz $S_1 = \{a_1 < a_2 < ... < a_k\},\ S_2 = \{b_1 < b_2 < ... < b_l\}$, akkor $k + l - 1$ összehasonlítás elég, hiszen ha az új halmazba a legkisebbtől sorban rakjuk be az elemeket, mindig legfeljebb 2 jöhet szóba: a két eredeti halmaz legkisebb, még fel nem használt eleme, az utolsó berakásánál viszont nem kell összehasonlítani.

Térjünk vissza egy adott halmaz rendezésére! Az n elemű halmazban először 1-1 elemet fésülünk össze, utána a párokat, és így tovább. Nyolc elemmel például:

(4- 7), (5- 3), (9- 2), (1- 0) ® (4, 7- 3, 5), (2, 9- 0, 1) ® (3, 4, 5, 7- 0, 1, 2, 9) ® (0, 1, 2, 3, 4, 5, 7, 9).

Az első összefésülés sorozatban 4 összefésülés volt, mindegyik 1 mérést igényelt. A második összefésülés sorozatban 2 összefésülés volt, mindegyik 3 méréssel. Az utolsó összefésülés sorozat csak 1 összefésülésből áll, de annak végrehajtásához 7 mérés is kellett.

Az így használt összehasonlítások számát könnyű kiszámolni, ha n kettőhatvány: $n = 2^i$, azaz $i=\log_2 n$:

$$1\times\frac{n}{2}+3\times\frac{n}{4}+...+\left(2^i-1\right)\times\frac{n}{2^i}=\left(n-\frac{n}{2}\right)+\left(n-\frac{n}{4}\right)+...+\left(n-\frac{n}{2^i}\right)=i\times n-\left(\frac{n}{2}+\frac{n}{4}+...+\frac{n}{2^i}\right)=i\times n-(n-1).$$

Ez tehát $(n\times\log_2 n - n + 1)$ összehasonlítás. Ha n nem kettőhatvány, akkor $n\times\lceil\log_2 n\rceil$ felülbecsüli az összehasonlítások számát ($\lceil x\rceil$ az x szám felső egészrészét jelöli, pl. $\lceil 3{,}14\rceil = 4$).

Ez a módszer nagyon erős, azt lehet mondani, hogy – a mozgatásokat is figyelembe véve – körülbelül $2n\log_2 n$ lépésből áll.

Öt elem rendezésére az összes előbbinél gyorsabb eljárást találtunk (5golyo.html 3. módszer). Sokkal jobb eljárás azonban nem létezhet.

Ha csupa különböző elemek vannak, akkor a lehetséges sorrendek száma $n!$, és minden összemérés az eseteket két komplementer részre osztja: az esetek egyik részében az egyik mért dolog lesz a nagyobb, az összes többi részben a másik. Így a szóbajövő esetek száma − a barkochbához hasonlóan − szerencsét nem feltételezve csak feleződhet, ezért a rendezéshez $\log_2 n!$ összehasonlításra szükség lesz, ami nagyságrendileg $n\times\log_2 n$-nel egyenlő. Magyarázatként lásd az 1. és a 3. Megjegyzést.

1. Megjegyzés (algoritmusok sebessége)

Az algoritmusok sebességét tipikusan úgy jellemzik, hogy egy rögzített skálához hasonlítják. A skála alapjául néhány jól ismert függvényt vesznek, pl. $g_0(n) = 1$, $g_1(n) = \log_2 n$, $g_2(n) = n$, $g_3(n) = n\cdot\log_2 n$, $g_4(n) = n^2$, …, $g_5(n) = 2^n$, … és az algoritmusokat ezekhez mérik. Ha egy feladat megoldásához n adattal, egy adott módszerrel maximálisan $f(n)$ művelet elvégzése szükséges, és $g(n)$ egy olyan jól ismert függvény, melyre $\frac{f(n)}{g(n)}$ az n növekedésével egy konstans korlát alatt marad, akkor azt mondjuk, hogy a módszer nagyságrendileg $g(n)$ műveletet használ, röviden $f(n) = O(g(n))$. (Ejtsd „ordó géen”!)

2. Megjegyzés (A 2/1.-2/3. algoritmusok sebessége)

A buborék-módszer sebessége tehát $O(n^2)$, a beszúrásos rendezésé $O(n\times\ln n)$ és megmutatható, hogy az összefésüléses rendezésé is $O(n\times\ln n)$.

3. Megjegyzés (A rendezési feladat minimális lépésszámáról)

Tegyük fel, hogy k összehasonlítással bármely n számot sorba lehet rakni! Ekkor vegyünk n számot, nézzük mind az $n!$ permutációját. Az eljárásunk k bit bemenő adatból legfeljebb 2^k különböző eredményt generálhat, ezért

$2^k \le n! \Rightarrow k \le \log_2 n!$. A Stirling- formulát felhasználva elég nagy n-re ez legalább

$$n\log_2 n! \ge n\log_2 \left(\frac{n}{e}\right)^n \sqrt{2\pi n} = n(\log_2 n - \log_2 e) + \frac{1}{2}(\log_2 n + \log_2 2\pi) = O(n\log_2 n).$$

Ajánló
Mathworld enciklopédia a Stirling formuláról: http://mathworld.wolfram.com/StirlingsApproximation.html
Lóczi Lajos: A faktoriális alsó és felső becslései, http://www.math.bme.hu/~jtoth/FelsMma/faktorialis.ps

4. Megjegyzés (bináris keresés)
A fenti vizsgálatokban feltettük, hogy a golyók különböző tömegűek (ill. az összehasonlítandó számok páronként különbözőek). Ha megengednénk azonos tömegeket (egyenlő számokat), akkor az összehasonlító mérésnek nem kétféle, hanem háromféle eredménye is lehetne: egyik nagyobb, másik nagyobb, egyenlő. Ez megfelelő körülmények között jelentősen befolyásolhatja a mérések számát.

3. feladat (Leghosszabb növő sorozat keresése)
Keressük $S = (a_1, a_2, ..., a_n)$ sorozatban minél hosszabb $(a_{i(1)}, a_{i(2)}, ..., a_{i(k)})$ részsorozatát! Tehát k-t kell maximalizálni úgy, hogy az s, t indexek bármely $1 \le s < t \le k$ értékeire teljesüljön az $i(s) < i(t)$ és az $a_{i(s)} < a_{i(t)}$ egyenlőtlenség is. (Feltehetjük, hogy a sorozatban nincs két egyenlő elem.)

Például a (3, 7, 5, 8, 4, 11)-ben a (3, 5, 8, 11) egy 4 hosszú növekvő részsorozat, de van-e hosszabb?

3/1. Módszer (Naiv algoritmus)
Egy másik sorozatot definiálunk: *h(i)*= az *i*. elemmel végződő leghosszabb, növekvő részsorozat hossza. Először *h(1)*-et számoljuk ki, utána *h(2)*-t, és az egyre nagyobb indexűeket.

Nyilván $h(1)=1$. Ha $1<j$, akkor $h(j)$ értéke a következőképpen számolható ki: Végignézzük a sorozat j-edik elemét megelőző elemek közül azokat, amelyek a j-edik elemnél kisebbek, és fölírjuk mindezen elemekhez írt h értékeket. A $h(j)$ érték ezen maximumnál eggyel nagyobb, illetve $h(j) = 1$, ha nem is volt nála kisebb elem. A leghosszabb növő részsorozat hossza az új sorozat (a h-sorozat) értékeinek maximuma.

Például:

4	3	7	5	6
1				
1	1			
1	**1**	2		
1	**1**	2	2	
1	**1**	2	**2**	3

Itt a leghosszabb növő részsorozat tehát 3 hosszú.

Eddig csak a sorozat hosszával törődtünk, szerencsére a h sorozatból visszafejthető a(z egyik) leghosszabb sorozat is. Kiválasztjuk a(z egyik) maximális végét, megkeressük az előtte levő kisebbek közül egyet, amelyiknek h-ja pont egyel kisebb, így haladhatunk visszafelé.

Itt minden számot össze kellett hasonlítani az összes nála kisebbel, hogy megnézhessük a h-jukat. Ez $O(n^2)$ lépés. A visszafejtés lineáris, tehát cn^2 összehasonlítás kell, az algoritmus $O(n^2)$ sebességű.

A következő fejezetben ennél gyorsabb algoritmust találunk a 3. feladat megoldására.

2. Tablók

Tabló:
Egy olyan számtáblázat, melyben
- egész számok szerepelnek
- a táblázat sorai balra vannak igazítva
- a sorok hosszai (a benne lévő számok száma) fentről lefelé nem nőnek
- a számok jobbra és lefelé szigorúan nőnek

Az *i*. sorban szereplő számok száma λ_i. Ha a tablónak *m* sora van és a sorok hossza rendre $\lambda_1,\lambda_2,...,\lambda_m$, akkor azt mondjuk, hogy a tabló típusa $\lambda=\{\lambda_1,\lambda_2,...,\lambda_m\}$. Ha a tabló *n* darab számból áll, $\sum_{i=1}^{m}\lambda_i=n$, akkor ezt a jelölést alkalmazzák: $\lambda\vdash n$. Ebben a cikkben technikai okokból helyenként a $\lambda\neg n$ jelölést használjuk.
Ha egy tablóra $\lambda\vdash n$ és a benne szereplő elemek az 1, 2, ..., *n* számok a tablót standard tablónak hívjuk.

Például:

1	2	4	7	9
3	5	6	8	
10	12	13	15	
11	14			

Itt $\lambda_1=5,\lambda_2=4,\lambda_3=4,\lambda_4=2,\lambda=(5,4,4,2)$, $\lambda\vdash n$, ahol *n*=15.

4. feladat

Soroljuk fel az összes **a)** 2-elemű, **b)** 3-elemű, **c)** 4-elemű standard tablót!

(Tehát írjuk be az 1, 2, 3, 4 számokat az összes lehetséges módon tablókba.)

A 4. feladat megoldása
Felsoroljuk a lehetséges λ típusokat és mindegyikhez felsoroljuk az olyan típusú tablókat. Alább f_λ jelöli a λ típusú standard tablók számát.

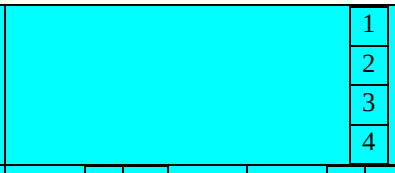
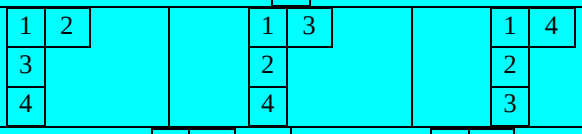
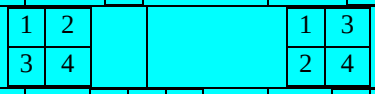
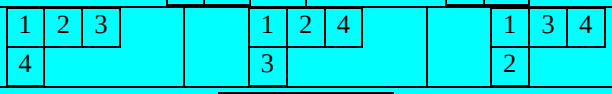
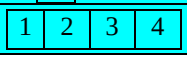
a)

λ = (1, 1)	1 2	$f_\lambda = 1$
λ = (2)	1 2	$f_\lambda = 1$

b)

λ = (1, 1, 1)	1 2 3	$f_\lambda = 1$
λ = (2, 1)	1 2 3 ; 1 3 2	$f_\lambda = 2$
λ = (3)	1 2 3	$f_\lambda = 1$

c)

l = (1, 1, 1, 1)		$f_l = 1$
l = (2, 1, 1)		$f_l = 3$
l = (2, 2)		$f_l = 2$
l = (3, 1)		$f_l = 3$
l = (4)		$f_l = 1$

5. Megjegyzés (az f_l értékek négyzetösszege)

Képezzük az egyes típusokhoz kapott végeredmények (a jobb oldali oszlopokban található számok) négyzetösszegét!

a) $1^2 + 1^2 = 2$, **b)** $1^2 + 2^2 + 1^2 = 6$, **c)** $1^2 + 3^2 + 2^2 + 3^2 + 1^2 = 24$.

A kapott eredmények ismerős számok. Ezt fogalmazza meg az alábbi 1. Tétel.

6. Megjegyzés (az f_l értékek összege)

Most képezzük az egyes típusokhoz kapott végeredmények összegét!

a) $1 + 1 = 2$, **b)** $1 + 2 + 1 = 4$, **c)** $1 + 3 + 2 + 3 + 1 = 11$.

A kapott eredmények kevésbé ismerősek, de nevezetesek. Ezzel kapcsolatos a későbbi 6. feladat.

1. Tétel

Ha f_l a l típusú, standard tablók száma, akkor $\sum_{l \vdash n} (f_l)^2 = n!$

Tulajdonképpen ezt fogjuk belátni a következőkben.

Robinson- Schensted-algoritmus

Adott egy *t* tabló, egy *x* egész, ami nem eleme a tablónak. Készítünk egy *t'* tablót, aminek *x* is eleme. Az eljárás a következő:

Végigmegyünk *x*-el az első soron. Ha találunk nála nagyobb elemet, azt „kilökjük", ha nem beillesztjük a sor végére. Ha van kilökött elem, azzal a következő soron megyünk végig.

Például, ha a tabló a következő:

1	2	4	8
6	7		
9			

a beszúrandó elem pedig 3, akkor az algoritmus a következőt csinálja:

1	2	**3**,4	8
6	7		
9			

®

1	2	3	8
4,6	7		
9			

®

1	2	3	8
4	7		
6,9			

®

1	2	3	8
4	7		
6			
9			

Meg kell mutatni, hogy ez az algoritmus tényleg tablót eredményez. Nem romlik-e el valamelyik tulajdonság?

A sorokban nyilván jó marad, hiszen vagy a legvégére rakunk be egy nagy elemet, vagy egyet kicserélünk egy kisebbre, de az előzőnél nagyobbra.
Mivel kisebbre cseréltünk le egy elemet, ha van alatta levő, annál kisebb marad. Nem lehet az sem, hogy a kicserélt elem a fölötte levőnél kisebb legyen, hiszen ha kilökünk egy elemet a következő sorban legfeljebb eddig a pontig megyünk (a kilökött elem alatt nála nagyobb szám, vagy üres hely van).
Végül pontosan egy sor hosszát növeltük eggyel (vagy egy új sort nyitottunk). Ezt a helyet nevezzük az új helynek. A fönti példában ez a 9-es helye (4. sor, 1. oszlop).

Ajánló:
Az algoritmus interaktív változata: http://www.math.uconn.edu/~troby/Goggin/BumpingAlg.html
Wikipedia a Robinson- Schensted-algoritmusról: http://en.wikipedia.org/wiki/Robinson-Schensted_algorithm

5. feladat

Mutassuk meg, hogy *t'* és az új hely ismeretében kitalálhatjuk, hogy mi volt az eredeti tabló, és melyik a beszúrt elem!

Megoldás (5. feladat)

Ha az első sorban van az új hely, akkor azt az elemet szúrtuk be, ami ott volt. Különben az új helyen lévő elemet az előző sorban lévő legnagyobb, nála kisebb elemnek kellett kiütnie. Ez a kiütött elemre is igaz.

Nem minden sor utolsó eleme lehet új hely, ha a következő sor ugyanolyan hosszú, a visszafejtés után kapott számtáblázat nem tabló, hiszen a sorok hossza lefelé van, ahol csökken. Különben viszont tablót kapunk, ami az előbbihez hasonlóan belátható.

Ennek felhasználásával fogjuk belátni az 1. tételt.

Bizonyítás (1. tétel)

Legyen $p = x_1, x_2, ..., x_n$ az 1, 2, ..., *n* számok egy permutációja. Definiálunk 2 tablót:
- t_p az a tabló, amit az üres tablóból indulva a Robinson-Schensted algoritmussal az $x_1, x_2, ..., x_n$ számok ilyen sorrendben beszúrásával kapunk.
- q_p pedig ugyanezzel az algoritmussal az, hogy az új helyek milyen sorrendben keletkeztek.

Nyilván $\lambda_{t_n} \vdash n,\ \lambda_{q_n} \vdash n$.

Például, ha $p = 2,3,1,5,4$, akkor

Beszúrt elem	2	3	1	5	4
t_p	2	2 3	1 3 2	1 3 5 2	1 3 4 2 5
q_p	1	1 2	1 2 3	1 2 4 3	1 2 4 3 5

A permutációk és a tabló- párok kölcsönösen és egyértelműen megfelelhetők egymásnak (az 5. feladat miatt). λ típusú tabló-párból f_λ^2 van, ezért $\sum_{\lambda \vdash n} (f_\lambda^2) = n!$, ami a tétel állítása.

6. feladat

Mutassuk meg, hogy $\sum_{\lambda \vdash n} f_\lambda$ megegyezik *n* elem másodrendű permutációinak számával! A másodrendű permutációk azok a permutációk, amelyeket kétszer végrehajtva az identitást kapjuk.

Például az (1, 5, 3, 7, 2, 6, 4) másodrendű permutáció: az 1, 3, 6 végig helyben marad, a 2, 5 és 4, 7 párok elemei felcserélődnek, kétszeri végrehajtás után eredeti helyükre térnek vissza. Az (1, 5, 2, 7, 3, 6, 4) viszont nem másodrendű, hiszen kétszer végrehajtva az (1, 5, 2, 4, 3, 6, 7) permutációt kapjuk.

A 6. feladat megoldása: Lásd http://matek.fazekas.hu/portal/eloadas/2005/6_feladat_erdelyi.html oldalunkat.

7. Megjegyzés: Igazolható, hogy f_{l} minden prímosztója legfeljebb *n*.

> **2. tétel (Schensted-tétel)**
> Legyen $\mathsf{p} = x_1, x_2, ..., x_n$ permutáció (nem feltétlen az (1, 2, 3,..., n)-é tulajdonképpen tetszőleges sorozat, aminek nincs 2 ugyanolyan eleme), t_{p} a tablója. Ekkor l_1 (a tabló első sorának hossza) a p-ben található, leghosszabb növő részsorozat hossza.

A Schensted-tétel bizonyítása

Képzeljük el az t_{p} tabló felépítésének folyamatát a Robinson-Schensted algoritmus szerint. Álljunk meg akkor, amikor az x_k elem éppen bekerül a tablóba (feltétlenül az első sorba). Legyen $H(k)$ annak az oszlopnak a sorszáma, ahova x_k épp most bekerült (lehet, hogy később egy másik sor másik oszlopában fejezi be a teljes algoritmust).

Állításunk az, hogy $H(k)=h(k)$, *h* a 3/1. módszerben használt jelölés, az x_k-val végződő leghosszabb sorozat hossza.

Ezt teljes indukcióval látjuk be (*n*-re, a sorozat hosszára). *n*= 1-re az állítás nyilván igaz, a leghosszabb növő sorozat 1 hosszúságú, az első elem is az első oszlopba kerül.

Tegyük fel, hogy állításunk *n*=*s*-ig igaz, kíséreljük meg igazolni *n*=*s*+1-re! Ha a $\mathsf{p}=(x_1, x_2, ..., x_s, x_{s+1})$ permutációra futtatjuk a Robinson- Schensted-algoritmust, de megállunk x_{s+1} érkezése előtt, akkor eddig a $\mathsf{p}'=(x_1, x_2, ..., x_s)$ permutációra futtattuk le az algoritmust, létrehoztuk a $t_{\mathsf{p}'}$ tablót. Indukciós feltételünk szerint a *H*(1), *H*(2), ..., *H*(*s*) számok megegyeznek a *h*(1), *h*(2), ..., *h*(*s*) számokkal.

A *h*(1), *h*(2), ..., *h*(*s*) számok nem feltétlenül különbözők, tehát egy konkrét *h** érték az 1, 2, ..., *s* indexek közül többhöz is tartozhat. Ez azt jelenti, hogy az $x_1, x_2, ..., x_s$ közül több olyan is lehet, amely egy *h** hosszú növő sorozat utolsó eleme, de nem utolsó eleme *h**-nál hosszabb növő sorozatnak. Pl. indukciós feltevésünk miatt a $t_{\mathsf{p}'}$ tabló első sorának *h**-adik oszlopában szereplő x_k számra is *h*(*k*)= *h**, hiszen a Robinson- Schensted-algoritmus során az elemek soron belül nem változtatják a helyüket. Az is igaz, hogy ez az x_k a legkisebb olyan szám $x_1, x_2, ..., x_s$ közül amelynek „*h*-ja” *h**, hiszen az első sor *h**-adik helyéről mindegyiket egy nála kisebb ütötte ki.

A $t_{\mathsf{p}'}$ tablóba most beillesztjük az x_{s+1} elemet. Ez az első sorba kerül, mondjuk a *h**-adik helyre, és ha ez nem egy új hely, akkor kilök onnan egy elemet. Azt kell megmutatnunk, hogy $h(x_{s+1})=h^*$, azaz nem létezik *h**-nál hosszabb x_{s+1}-re végződő növő sorozat, de *h** hosszú sorozat létezik.

Ha lenne legalább *h**+1 hosszú növekvő sorozat, aminek utolsó eleme x_{s+1}, akkor annak *h**-adik eleme legalább akkora lenne, mint a $t_{\mathsf{p}'}$ tabló első sorának *h**-adik eleme, aminél viszont kisebb az x_{s+1} elem, hiszen azt ütötte ki. Ez ellentmondás.

Konstruálunk *h** hosszú x_{s+1}-re végződő növő sorozatot. A sorozat tagjait visszafelé haladva képezzük. Az utolsó, tehát a *h**-adik tag nyilván x_{s+1}. Ha a sorozat *i*-edik tagja már megvan, akkor az (*i*- 1)-edik tag legyen egy olyan elem, amely a Robinson- Schensted-algoritmust az első sor (*i*-1)-edik helyén kezdte, és még az első sorban volt, amikor a sorozat *i*-edik tagja belépett (az első sor *i*-edik helyére). A konstrukciós lépés biztosítja, hogy sorozatunk megelőző tagja az utóbbi tagot nagyságrendben és a p permutációban is megelőzi.

Például ha a sorozat (4, 3, 7, 5, 6) a tabló első sora így alakul: (**4**), (**3**), (3, **7**), (3, **5**), (3, 5, **6**). Így a *H* értékei: 1, 1, 2, 2, 3. Ezt kaptuk a 3/1. módszerben ugyanerre a sorozatra *h*-nak.

Ezzel viszont gyorsabb algoritmust kapunk a leghosszabb növő részsorozat megtalálására (3. feladat)!

3/2. módszer

Az előbb gyártott tablót, illetve annak az első sorát vizsgáljuk. Az elemeket bináris kereséssel illesztjük be, tehát legfeljebb $\log_2 n+1$ összehasonlítással, összesen nagyságrendileg $n(\log_2 n+1)$-nel.

8. Megjegyzés

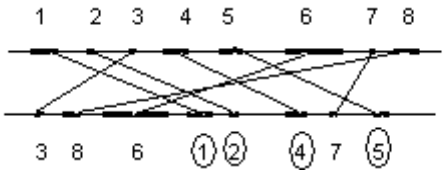
Nem biztos, hogy a tabló első sora egy növő részsorozat, hiszen az ott előforduló elemek indexei nem biztos, hogy balról jobbra nőnek. Például a (2, 3, 1) sorozatra a kapott első sor (1, 3).

A 3/2 módszert a genetikában is használják.

A genetikusok manapság már képesek felsorolni az élőlények génjeiben található kódsorozatot. Mit jelentenek az egyes kódrészletek? Milyen rokonságban állnak a különböző élőlények? Mindkét kérdés tisztázásában nagy jelentőséggel bír, ha sikerül megfeleltetni egymásnak a különböző egyedek, sőt különböző fajok kromoszómáinak egymásnak megfelelő vagy egymáshoz nagyon hasonló részeit.

Nehézséget okoz, hogy a genetikai kód rendkívül hosszú. Egy teljes, alapos összehasonlítás négyzetes lépésszámot igényelne, ez túl van a számítógépek kapacitásán. A „forró helyek” megkeresése, majd a fenti algoritmus alkalmazása megfelelő megoldást képes adni.

A „forró hely” (hot spot) a két kódban két egymással tökéletesen azonos, rövid (5-10 nukleotidából álló) részlet. Az algoritmus első lépésében megkeressük a két genetikai sorozatban a forró helyeket. Az első kódsorozatban ezeket besorszámozzuk, a másodikban a besorszámozott részletek azonosított párjaira is ráírjuk ugyanazt a sorszámot. Így, ha a második kódon végigmegyünk, akkor a hot spotot sorszámai nem alkotnak egy egyesével növekedő sorozatot, sőt nem is növekedő ez a sorozat. Itt segít korábbi algoritmusunk. Keresünk egy minél hosszabb növekedő részsorozatot. A két kódban ezen részsorozatnak megfelelő forró helyek úgy feleltethetők meg egymásnak, hogy a megfelelő párokat összekötő vonalak (lásd az ábrát) nem keresztezik egymást. Ez az alapja a két kódsorozat teljes megfeleltetésének.



A teljes megfeleltetés során a hot spotok közötti részeket is megpróbálják egymáshoz rendelni, párbaállítani. Ilyenkor előfordulhat az is, hogy az egyik gén egyik kódrészletének a másik génben egy sokkal rövidebb rész, esetleg a „semmi”, azaz egy szóköz felel meg. A különböző élőlények látható hasonlóságait és megfigyelhető különbségeit összevethetjük a DNS-láncban így megfigyelhető hasonlóságokkal, különbségekkel. Ez a megközelítés fontos szerepet játszik a genetikai kód értelmezésére, megértésére irányuló törekvésekben.

3. További tételek

Az alábbi néhány nevezetes tétel szorosan összefügg eddigi vizsgálatainkkal.

Greene-tétel
Legyen p permutáció, t_p az 1.tétel bizonyításában definiált tabló, típusa $\lambda = \{\lambda_1, \lambda_2, ..., \lambda_m\}$. A p sorozat azon leghosszabb részsorozatának elemszáma, amely felbomlik j darab növő részsorozat uniójára $\lambda_1 + \lambda_2 + ... + \lambda_j$.

Példa
A p=(2, 8, 3, 4, 1, 5, 7, 6, 10, 9) permutáció esetén:

t_p =

1	3	4	5	6	9
2	7	10			
8					

Ezek szerint p.nek van egy 6-elemű és egy attól diszjunkt 3 elemből álló részsorozata. Lásd a vastagon szedett és az aláhúzással jelölt részsorozatot: (**2**, 8, **3**, **4**, 1, **5**, 7, **6**, 10, **9**).

Permutációk fogyó részsorozatai
A p permutációban a leghosszabb fogyó részsorozat hossza megegyezik a t_p tabló sorainak számával, azaz m-mel, ha t_p típusa $\lambda = \{\lambda_1, \lambda_2, ..., \lambda_m\}$.

Erdős- Szekeres-lemma
Ha az n elemű sorozatban a leghosszabb növő részsorozat hossza N, a leghosszabb fogyóé F, akkor $NF \geq n$.

Ezt a már sokszor használt tablónk láthatóvá teszi, az egyenlőség is szemléletes: amikor tabló téglalap alakú.

Mi permutációkkal dolgoztunk, de nem nehéz kiterjeszteni az állításainkat általános sorozatokra (ahol lehet néhány ugyanolyan elem). Az így kapott tablók félig standard, vagy Young-tablók lesznek. A különbség csak annyi, hogy a sorokban az elemek nem szigorúan monoton nőnek, hanem helyenként egyenlő elemek is lehetnek., ugyanígy a leghosszabb növő sorozatokban is lehetnek egyenlő számok. A Robinson- Schensted-megfeleltetés kiterjeszthető a permutációkról a sorozatokra, $n!$-ról n^n -re.

Ajánló
Az Erdős- Szekeres-lemma egy másik bizonyítása: http://www.math.bme.hu/~biro/zh/node3.html