Algoritmusok tervezése és elemzése

2021. ősz

Lehetetlenségi tételek

Arrow lehetetlenségi tétele

Azt vizsgáljuk, hogy létezik-e olyan általános algoritmus, amely megnyugtató módon képes megoldani az egyéni preferenciák közösségivé való aggregálását.

Legyen $A = \{a_1, a_2, \ldots, a_m\}$ különböző alternatívák egy tetszőleges halmaza. Legyen továbbá $E = \{e_1, e_2, \ldots, e_n\}$ egyének egy halmaza, akik közül mindenkinek van egy teljes, reflexív, antiszimmetrikus és tranzitív preferencia-rendezése az alternatívák A halmazán. Jelölje ezeket a preferencia-rendezéseket $\succcurlyeq_1, \succcurlyeq_2, \ldots, \succcurlyeq_n$, itt $a_i \succ_t a_j$ azt jelenti, hogy az e_t egyén az a_i alternatívát preferálja az a_j alternatívával szemben. Megengedjük, hogy egy egyén egyformán preferáljon két alternatívát. Vezessük még be a $\succcurlyeq = (\succcurlyeq_1, \succcurlyeq_2, \ldots, \succcurlyeq_n)$ jelölést, a $\succcurlyeq$ rendezett sorozatot preferencia-profilnak fogjuk nevezni.

Egy az összes lehetséges $\succcurlyeq = (\succcurlyeq_1, \succcurlyeq_2, \ldots, \succcurlyeq_n)$ preferencia-profilon értelmezett $\mathcal{R}$ függvényt, amely minden $\succcurlyeq$ preferencia-profilhoz az A alternatíva halmaz egy teljes, reflexív, antiszimmetrikus és tranzitív $\trianglerighteq$ preferencia-rendezését rendeli, társadalmi jóléti függvénynek nevezünk.

Egy társadalmi jóléti függvényre viszonylag természetes megkövetelni a következő tulajdonságok teljesülését.

- *Egybehangzó vélemények érvényesülésének elve:* Legyenek a_i és a_j tetszőleges alternatívák. Ha egy $\succcurlyeq = (\succcurlyeq_1, \succcurlyeq_2, \ldots, \succcurlyeq_n)$ preferencia-profilban $a_i \succ_t a_j$ minden $1 \leqslant t \leqslant n$ esetén, akkor $a_i \triangleright a_j$.

- *Irreleváns alternatíváktól való függetlenség elve:* Legyenek a_i és a_j tetszőleges alternatívák, valamint legyenek $\succcurlyeq = (\succcurlyeq_1, \succcurlyeq_2, \ldots, \succcurlyeq_n)$ és $\succcurlyeq' = (\succcurlyeq'_1, \succcurlyeq'_2, \ldots, \succcurlyeq'_n)$ tetszőleges preferencia-profilok. Ha minden $1 \leqslant t \leqslant n$ esetén az e_t egyén preferenciája a_i és a_j vonatkozásában a $\succcurlyeq$ és $\succcurlyeq'$ preferencia-profilokban ugyanaz, akkor a_i és a_j egymáshoz képesti pozíciója az $\trianglerighteq = \mathcal{R}(\succcurlyeq)$ és $\trianglerighteq' = \mathcal{R}(\succcurlyeq')$ preferencia-rendezésekben ugyanaz.

- *Diktatúra hiánya:* Nincs olyan e_t egyén, hogy tetszőleges $\succcurlyeq = (\succcurlyeq_1, \succcurlyeq_2$

$, \ldots, \succcurlyeq_n)$ preferencia profilra $a_i \succ_t a_j$ mindig maga után vonná $a_i \rhd a_j$ teljesülését.

Arrow megmutatta, hogy amennyiben az alternatívák száma legalább három, nem létezik olyan társadalmi jóléti függvény, amelyre a fenti három feltétel egyidejűleg teljesülne!

Arrow-tétel. Ha az alternatívák száma legalább három, és egy társadalmi jóléti függvényre teljesül az egybehangzó vélemények érvényesülésének elve és az irreleváns alternatíváktól való függetlenség elve, akkor az szükségképpen diktatórikus.

Bizonyítás. Legyen az alternatívák száma legalább három, és tekintsünk egy olyan $\mathcal{R}$ társadalmi jóléti függvényt, amelyre teljesül az egybehangzó vélemények érvényesülésének elve és az irreleváns alternatíváktól való függetlenség elve.

Legyenek a_i és a_j tetszőleges alternatívák, és tekintsünk egy olyan $\succcurlyeq = (\succcurlyeq_1, \succcurlyeq_2, \ldots, \succcurlyeq_n)$ preferencia-profilt, amelyben $a_i \succ_t a_j$ minden $1 \leqslant t \leqslant n$ esetén. Cseréljük fel egymás után az a_i és a_j alternatívákat a $\succcurlyeq_t$ preferencia-rendezésekben. Az egybehangzó vélemények érvényesülésének elve szerint az első csere előtt $a_i \rhd a_j$, míg az utolsó csere után $a_j \rhd a_i$ teljesül. Jelölje $e_{t_{ij}}$ az első olyan egyént, akinél a_i és a_j cseréje után $a_i \rhd a_j$ már nem áll fenn. Az irreleváns alternatíváktól való függetlenség elvével összhangban $e_{t_{ij}}$ független $\succcurlyeq$-től.

Legyenek ezután a_i, a_j és a_k tetszőleges alternatívák, és tekintsünk egy olyan $\succcurlyeq'$ preferencia-profilt, amelyben a_i, a_j és a_k egymáshoz képesti pozíciói a következők:

e_1	$\cdots$	$e_{t_{ij}-1}$	$e_{t_{ij}}$	$e_{t_{ij}+1}$	$\cdots$	e_n
a_j	$\cdots$	a_j	a_i	a_i	$\cdots$	a_i
a_k	$\cdots$	a_k				
a_i	$\cdots$	a_i	a_j	a_j	$\cdots$	a_j
			a_k	a_k	$\cdots$	a_k

Ekkor $\mathcal{R}(\succcurlyeq')$-ben $a_i \rhd a_j \rhd a_k$. Itt az első reláció $e_{t_{ij}}$ definíciójából adódik, a második pedig az egybehangzó vélemények érvényesülésének elvéből következik.

Tekintsünk ezután egy olyan $\succcurlyeq''$ preferencia-profilt, amelyben a_i, a_j és a_k egymáshoz képesti pozíciói a következők (a négyzet a_k lehetséges pozícióit jelöli a másik két alternatívához képest, ne feledjük, holtverseny lehetséges):

e_1	$\cdots$	$e_{t_{ij}-1}$	$e_{t_{ij}}$	$e_{t_{ij}+1}$	$\cdots$	e_n
$\square$	$\cdots$	$\square$				
$\square\, a_j$	$\cdots$	$\square\, a_j$	a_j	a_i	$\cdots$	a_i
$\square$	$\cdots$	$\square$		$\square$	$\cdots$	$\square$
a_i	$\cdots$	a_i	a_i	$\square\, a_j$	$\cdots$	$\square\, a_j$
			a_k	$\square$	$\cdots$	$\square$

Ekkor $\mathcal{R}(\succcurlyeq'')$-ben $a_j \trianglerighteq a_i \rhd a_k$. Az első reláció itt is $e_{t_{ij}}$ definíciójából adódik, a második pedig az irreleváns alternatíváktól való függetlenség elvéből következik. Valóban, az egyéni preferenciák a_i és a_k vonatkozásában megegyeznek a $\succcurlyeq'$ és $\succcurlyeq''$ profilokban.

Ez viszont az irreleváns alternatíváktól való függetlenség elvével összhangban azt jelenti, hogy $a_j \succ_{t_{ij}} a_k$ maga után vonja $a_j \rhd a_k$ teljesülését (a többi egyén preferenciája a_j és a_k vonatkozásában nem számít).

Ebből következik, hogy az $e_{t_{jk}}$ egyént definiáló felcserélési eljárásban $a_j \rhd a_k$ nem változhat addig, amíg az $e_{t_{ij}}$ egyén a_k elé sorolja a_j-t, így $t_{jk} \geqslant t_{ij}$. Másrészt az $e_{t_{kj}}$ egyént definiáló felcserélési eljárásban $a_j \rhd a_k$ biztos teljesül miután az $e_{t_{ij}}$ egyén a_k elé sorolja a_j-t, így $t_{kj} \leqslant t_{ij}$. Ennélfogva $t_{jk} \geqslant t_{ij} \geqslant t_{kj}$.

Az a_j és a_k alternatívák szerepét felcserélve, hasonlóan adódik, hogy $t_{kj} \geqslant t_{ik} \geqslant t_{jk}$. Következésképpen $t_{jk} = t_{ij} = t_{ik} = t_{kj}$. És mivel ez bármely három különböző a_i, a_j és a_k alternatívára teljesül, ezért az $e_{t_{ij}}$ egyén az i és j indexektől függetlenül mindig ugyanaz, és az $\mathcal{R}$ társadalmi jóléti függvény minden preferencia-profilhoz ennek az egyénnek a preferencia-rendezését rendeli, vagyis diktatórikus.

Kleinberg lehetetlenségi tétele

Mutatunk még egy ilyen jellegű, negatív eredményt. Legyen X tetszőleges nem üres halmaz, és legyen k egy pozitív egész szám. Az X halmaz k nem üres részhalmazának egy

$$\mathcal{C} = \{C_1, C_2, \ldots, C_k\}$$

családját az X egy k-klaszterezésének nevezzük, ha

- $C_1 \cup C_2 \cup \cdots \cup C_k = X$,
- $C_i \cap C_j = \emptyset$, ha $i \neq j$.

Azt mondjuk, hogy $\mathcal{C}$ egy klaszterezése X-nek ha $\mathcal{C}$ egy k-klaszterezése X-nek valamely pozitív egész k esetén. Egy $\mathcal{C}$ klaszterezésben szereplő halmazokra klaszterekként fogunk hivatkozni. Egy klaszterezést triviálisnak nevezünk, ha minden klaszter egyetlen elemből áll, vagy minden elem ugyanahhoz a klaszterhez tartozik.

Legyen $\mathcal{C}$ egy klaszterezése X-nek, továbbá legyenek $x, y \in X$. Azt fogjuk írni, hogy $x \sim_{\mathcal{C}} y$, ha x és y a $\mathcal{C}$ ugyanabban a klaszterében vannak, illetve $x \not\sim_{\mathcal{C}} y$ ha x és y a $\mathcal{C}$ különböző klasztereiben vannak.

Legyen X tetszőleges nem üres halmaz. Egy $d\colon X \times X \to \mathbb{R}_0^+$ függvényt távolságfüggvénynek nevezünk, ha bármely $x, y \in X$ esetén

- $d(x, y) = 0$ akkor és csak akkor, ha $x = y$,
- $d(x, y) = d(y, x)$.

Jegyezzük meg, hogy a háromszög-egyenlőtlenséget nem követeljük meg automatikusan!

Klaszterezés függvénynek nevezünk egy olyan függvényt, amely egy (X, d) párhoz, ahol X tetszőleges nem üres véges halmaz és d egy távolságfüggvény X-en, az X halmaz egy klaszterezését rendeli.

Egy klaszterezés függvényre viszonylag természetes megkövetelni a következő tulajdonságok teljesülését.

- Egy F klaszterezés függvényt skála-invariánsnak nevezünk ha tetszőleges (X, d) párra és λ pozitív számra $F(X, d) = F(X, \lambda d)$.

- Egy F klaszterezés függvényt konzisztensnek nevezünk, ha $F(X,d) = F(X,d')$ tetszőleges (X,d) párra, továbbá bármely olyan d' távolságfüggvényre az X-en, amelyre teljesül, hogy tetszőleges $x, y \in X$ esetén
  - $d'(x,y) \leqslant d(x,y)$ ha $x \sim_{\mathcal{C}} y$,
  - $d'(x,y) \geqslant d(x,y)$ ha $x \not\sim_{\mathcal{C}} y$,

  ahol $\mathcal{C} = F(X,d)$.
- Egy F klaszterezés függvényt teljesnek nevezünk, ha tetszőleges X halmazhoz és annak tetszőleges $\mathcal{C}$ klaszterezéséhez létezik olyan d távolságfüggvény X-en, hogy $F(X,d) = \mathcal{C}$.

Kleinberg megmutatta, hogy nem létezik olyan klaszterezés függvény, amelyre a fenti három tulajdonság egyidejűleg teljesülne!

Kleinberg-tétel. Nem létezik olyan klaszterezés függvény, amely skála-invariáns, konzisztens és teljes.

Bizonyítás. Kicsit többet fogunk bizonyítani. Legyen X tetszőleges nem üres halmaz, és legyenek $\mathcal{C}_1$ valamint $\mathcal{C}_2$ klaszterezései X-nek. Azt mondjuk, hogy $\mathcal{C}_1$ finomítása $\mathcal{C}_2$-nek ha bármely $\mathcal{C}_2$-beli klaszter előáll $\mathcal{C}_1$-beli klaszterek uniójaként. Megmutatjuk, hogy ha F skála-invariáns és konzisztens klaszterezés függvény, X legalább kételemű véges halmaz, és d_1 és d_2 távolságfüggvények X-en, akkor az $F(X,d_1)$ és $F(X,d_2)$ klaszterezések vagy megegyeznek, vagy egyik se finomítása a másiknak.

Indirekt tegyük fel, hogy valamilyen d_1 és d_2 távolságfüggvényekre $\mathcal{C}_1 = F(X,d_1)$ és $\mathcal{C}_2 = F(X,d_2)$ különböző klaszterezések, és mondjuk $\mathcal{C}_1$ finomítása $\mathcal{C}_2$-nek. Legyen

$$m_1 = \min\{d_1(x_i,x_j) \mid x_i \sim_{\mathcal{C}_1} x_j\},$$
$$M_1 = \max\{d_1(x_i,x_j) \mid x_i \not\sim_{\mathcal{C}_1} x_j\},$$

illetve

$$m_2 = \min\{d_2(x_i,x_j) \mid x_i \sim_{\mathcal{C}_2} x_j\},$$
$$M_2 = \max\{d_2(x_i,x_j) \mid x_i \not\sim_{\mathcal{C}_2} x_j\},$$

továbbá legyenek $a_1 < b_1$ és $a_2 < b_2$ olyan pozitív számok, hogy

$$a_1 \leqslant m_1,$$
$$b_1 \geqslant M_1,$$
$$a_2 \leqslant m_2,$$
$$b_2 \geqslant M_2.$$

Tekintsük most azt a d távolságfüggvényt X-en, amelyre $d(x_i, x_j) = \frac{a_1 a_2}{b_1}$ ha $x_i \sim_{\mathcal{C}_1} x_j$, továbbá $d(x_i, x_j) = a_2$ ha $x_i \sim_{\mathcal{C}_2} x_j$ de $x_i \not\sim_{\mathcal{C}_1} x_j$, végül $d(x_i, x_j) = b_2$ ha $x_i \not\sim_{\mathcal{C}_2} x_j$. Mivel F konzisztens, ezért $F(X, d) = F(X, d_2) = \mathcal{C}_2$.

Tekintsük ezután a $d' = \frac{b_1}{a_2} d$ távolságfüggvényt X-en. Mivel F skálainvariáns, ezért $F(X, d') = F(X, d) = \mathcal{C}_2$. Ugyanakkor $d'(x_i, x_j) = \frac{b_1}{a_2} \frac{a_1 a_2}{b_1} = a_1$ ha $x_i \sim_{\mathcal{C}_1} x_j$, valamint $d'(x_i, x_j) \geqslant \frac{b_1}{a_2} a_2 = b_1$ ha $x_i \not\sim_{\mathcal{C}_1} x_j$, így F konzisztens volta miatt $F(X, d') = F(X, d_1) = \mathcal{C}_1$, ami ellentmondás.

Legyen X tetszőleges nem üres véges halmaz, és legyen d egy távolságfüggvény X-en. Egy összekapcsolás-alapú klaszterező algoritmus először X minden elemét külön klaszterbe sorolja, aztán minden lépésben összekapcsolja egy klaszterré a két legközelebbi klasztert, amíg a klaszterek száma egy előre megadott, 1 és $|X|$ közötti értéket el nem ér.

Két klaszter távolságát egy összekapcsolás-függvény segítségével határozzuk meg. Ennek legelterjedtebb fajtái:

- minimális távolság szerinti: $\min_{x \in C_i, y \in C_j} d(x, y)$,
- átlagos távolság szerinti: $\frac{\sum_{x \in C_i, y \in C_j} d(x,y)}{|C_i||C_j|}$,
- maximális távolság szerinti: $\max_{x \in C_i, y \in C_j} d(x, y)$.

Az algoritmus megállási feltétele nem csak az lehet, hogy a klaszterek száma elért egy előre megadott értéket. Két másik lehetőséget említünk:

- A két legközelebbi klaszter távolsága nagyobb, mint αM, ahol M a maximális távolság az X elemei között, és $\alpha < 1$ adott pozitív szám.
- A két legközelebbi klaszter távolsága nagyobb, mint r, ahol r adott pozitív szám.

Állítás. A minimális távolság szerinti összekapcsolás algoritmus (klaszterezés függvény)

- eredeti verziója skála-invariáns, konzisztens, de nem teljes,
- második verziója $|X| \geqslant 3$ esetén skála-invariás, teljes, de nem konzisztens,
- harmadik verziója $|X| \geqslant 2$ esetén teljes, konzisztens, de nem skálainvariáns.

Oszd meg és uralkodj algoritmusok

Idézzük fel, hogy az összefésüléses rendezés alapötlete az volt, hogy először rendezzük külön-külön egy $A[1:n]$ tömb első felét és második felét, majd ezek tartalmát fésüljük össze. Az $A[1:n]$ tömb rendezésének feladatát tehát egy összefésülés árán két feleakkora résztömb rendezésére vezettük vissza. A résztömböket természetesen ugyanezen ötlettel rendeztük.

Ezt a (rekurzív) megközelítést oszd meg és uralkodj elvnek nevezik. Egy oszd meg és uralkodj algoritmus a rekurzió minden szintjén a következő három lépést hajtja végre.

(1) A feladatot több részfeladatra osztja, amelyek hasonlóak az eredeti feladathoz, de méretük kisebb.

(2) Rekurzív módon megoldja a részfeladatokat (ha a részfeladatok mérete elég kicsi, akkor közvetlenül oldja meg azokat).

(3) A részfeladatok megoldásait kombinálva előállítja az eredeti feladat megoldását.

Inverziók száma

Adott különböző számoknak egy $A[1:n]$ tömbje. Azt mondjuk, hogy valamely $1 \leqslant i < j \leqslant n$ esetén az (i, j) pár az A egy inverziója, ha $A[i] > A[j]$. Adjunk $O(n \log n)$ költségű algoritmust, amely meghatározza az $A[1:n]$ tömb inverzióinak számát!

Módosítsuk az összefésüléses rendezés algoritmust a következőképpen.

```
InverziókSzámaÖsszefésüléssel(A,p,q,r)
k=q-p+1
l=r-q
```

```
for i=1 to k do
  B[i]=A[p+i-1]
for j=1 to l do
  C[j]=A[q+j]
B[k+1]=INFINITY
C[l+1]=INFINITY
i=1
j=1
inv=0
for m=p to r do
  if B[i]<C[j]
    then
      A[m]=B[i]
      i=i+1
    else
      A[m]=C[j]
      inv=inv+k-i+1
      j=j+1
return inv

InverziókSzáma(A,p,r)
inv=0
if p<r then
  q=[(p+r)/2]
  inv=inv+InverziókSzáma(A,p,q)
  inv=inv+InverziókSzáma(A,q+1,r)
  inv=inv+InverziókSzámaÖsszefésüléssel(A,p,q,r)
return inv

InverziókSzáma(A,1,n)
```

Először az $A[1:n]$ tömböt az $A[1:q]$ és az $A[q+1:n]$ résztömbökre bontjuk, ahol $q = [(n+1)/2]$, és rekurzívan meghatározzuk az egyes résztömbök inverziószámát. Ezután következik azon (i', j') inverziók összeszámlálása, amelyekre $1 \leqslant i' \leqslant q < j' \leqslant n$. Ezt valósítja meg az `InverziókSzámaÖsszefésüléssel` eljárás. Jegyezzük meg, hogy mire az eljárás meghívásra kerül, az $A[1:q]$ és az $A[q+1:n]$ résztömbök már rendezettek.

Belátjuk az `InverziókSzámaÖsszefésüléssel` eljárás helyességét. Tegyük fel, hogy az eljárás során éppen az $A[p:q]$ résztömb $A[i']$ és az $A[q+1:r]$ résztömb $A[j']$ értékének összehasonlításánál tartunk, ahol $p \leqslant i' \leqslant q < j' \leqslant$

r. Azt is tegyük fel, hogy már minden olyan inverziót felderítettünk, amelynek az első komponense p és $i'-1$ között van, vagy a második komponense $q+1$ és $j'-1$ között.

- Ha $A[i'] < A[j']$, akkor $A[j'] < A[j'+1] < \cdots < A[n]$ miatt nincs olyan eddig nem látott inverzió, amelynek első komponense i', így ennél a lépésnél a felderített inverziók száma nem nő.
- Ha $A[i'] > A[j']$, akkor $A[q] > \cdots > A[i'+1] > A[i']$ miatt (i', j'), $(i'+1, j'), \ldots, (q, j')$ eddig nem látott új inverziók, így ennél a lépésnél a felderített inverziók száma $(q - i' + 1)$-gyel nő.

Innen az `InverziókSzámaÖsszefésüléssel` eljárás helyessége adódik.

Az algoritmus teljes költsége az összefésüléses rendezéséhez hasonlóan $O(n \log n)$.

Mester-módszer

Az oszd meg és uralkodj algoritmusok elemzését gyakran megkönnyíti a következő tétel.

Tétel. Legyen $T(n)$ a nemnegatív egész számok halmazán a

$$T(n) = aT(n/b) + f(n)$$

rekurzív egyenlettel definiált függvény, ahol $a \geq 1$ és $b > 1$ állandók, $f(n)$ pedig egy pozitív függvény (a rekurzív egyenletet tekintve itt kicsit pongyolán fogalmaztunk, igazából $T(n/b)$ helyett $T(\lfloor n/b \rfloor)$ vagy $T(\lceil n/b \rceil)$ írandó). Ekkor

- ha $f(n) = O(n^{\log_b a - \varepsilon})$ valamely $\varepsilon > 0$ állandóra, akkor
$$T(n) = \Theta(n^{\log_b a});$$
- ha $f(n) = \Theta(n^{\log_b a})$, akkor
$$T(n) = \Theta(n^{\log_b a} \log n);$$
- ha $f(n) = \Omega(n^{\log_b a + \varepsilon})$ valamely $\varepsilon > 0$ állandóra, továbbá
$$af(n/b) \leqslant cf(n)$$
valamely $c < 1$ állandóra minden elég nagy n esetén, akkor
$$T(n) = \Theta(f(n)).$$

Tegyük fel hogy egy algoritmus költségére a következő rekurzív képletet sikerült felállítanunk:

$$T(n) = 9T(n/3) + n.$$

Itt $a = 9$, $b = 3$ és $f(n) = n$. Mivel $n^{\log_b a} = n^{\log_3 9} = n^2$, továbbá $f(n) = O(n^{\log_3 9 - \varepsilon})$, ahol $\varepsilon = 1$, így alkalmazható a tétel első pontja, amely szerint

$$T(n) = \Theta(n^2).$$

Második példánk legyen a

$$T(n) = T(2n/3) + 1$$

rekurzív egyenlet. Itt $a = 1$, $b = 3/2$ és $f(n) = 1$. Most $n^{\log_b a} = n^{\log_{3/2} 1} = n^0 = 1$, így alkalmazható a tétel második pontja, amely szerint

$$T(n) = \Theta(\log n).$$

Harmadik példánk a

$$T(n) = 3T(n/4) + n \log n$$

egyenlet. Itt $a = 3$, $b = 4$, $f(n) = n \log n$ és $n^{\log_b a} = n^{\log_4 3} = O(n^{0{,}793})$. Mivel $f(n) = \Omega(n^{\log_4 3 + \varepsilon})$, ahol $\varepsilon = 0{,}2$, így alkalmazható a tétel harmadik pontja, ha a regularitási feltétel is teljesül. Most

$$af(n/b) = 3(n/4)\log(n/4) \leqslant (3/4)n \log n = cf(n),$$

ahol $c = 3/4$. Így

$$T(n) = \Theta(n \log n).$$

Végül vizsgáljuk meg a

$$T(n) = 2T(n/2) + n \log n$$

egyenletet. Itt $a = 2$, $b = 2$, $f(n) = n \log n$ és $n^{\log_b a} = n^{\log_2 2} = n$, így úgy tűnik a tétel harmadik esetével van dolgunk, hiszen $f(n) = n \log n$ aszimptotikusan nagyobb, mint $n^{\log_b a} = n$. Vegyük észre azonban, hogy nem polinomiálisan nagyobb, ugyanis

$$\frac{f(n)}{n^{\log_b a}} = \frac{n \log n}{n} = \log n$$

aszimptotikusan kisebb, mint n^ε bármely ε pozitív konstans esetén. Ezért ebben az esetben a tétel nem alkalmazható. Kifinomultabb technika felhasználásával belátható, hogy esetünkben

$$T(n) = \Theta(n \log^2 n).$$

Gyors mátrixszorzás

Adott két $n \times n$-es mátrix:

$$A = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{pmatrix} \quad \text{és} \quad B = \begin{pmatrix} b_{11} & b_{12} & \dots & b_{1n} \\ b_{21} & b_{22} & \dots & b_{2n} \\ \vdots & \vdots & & \vdots \\ b_{n1} & b_{n2} & \dots & b_{nn} \end{pmatrix}.$$

Ekkor a $C = AB$ szorzat egy szintén $n \times n$-es mátrix:

$$C = \begin{pmatrix} c_{11} & c_{12} & \dots & c_{1n} \\ c_{21} & c_{22} & \dots & c_{2n} \\ \vdots & \vdots & & \vdots \\ c_{n1} & c_{n2} & \dots & c_{nn} \end{pmatrix},$$

ahol

$$c_{ij} = a_{i1}b_{1j} + a_{i2}b_{2j} + \cdots + a_{in}b_{nj} \quad (1 \leqslant i, j \leqslant n).$$

Ha a kijelölt műveleteket a fenti módon végezzük el, ez n^3 elemi szorzást és $n^2(n-1)$ elemi összeadást igényel. Létezik ennél hatékonyabb eljárás?

Az egyszerűség kedvéért először tegyük fel, hogy n kettőhatvány. Ekkor az A, B, C mátrixok mindegyikét négy $n/2 \times n/2$-es méretű mátrixra bonthatjuk:

$$A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \qquad B = \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix} \qquad C = \begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix}$$

Vegyük észre, hogy a szorzatra vonatkozó összefüggések a részmátrixokkal is fennállnak:

$$\begin{aligned} C_{11} &= A_{11}B_{11} + A_{12}B_{21}, \\ C_{12} &= A_{11}B_{12} + A_{12}B_{22}, \\ C_{21} &= A_{21}B_{11} + A_{22}B_{21}, \\ C_{22} &= A_{21}B_{12} + A_{22}B_{22}. \end{aligned}$$

Az előbbi képletek mindegyike két $n/2 \times n/2$-es mátrixszorzást és a szorzatok összeadását tartalmazza.

Jelölje $S(n)$ két $n \times n$-es mátrix összeszorzásához szükséges elemi összeadások, $M(n)$ pedig az elemi szorzások számát. Mátrixszorzásunk költségére most a következő rekurzív képletek érvényesek:

$$\begin{aligned} S(n) &= 8S(n/2) + 4(n/2)^2, \\ M(n) &= 8M(n/2). \end{aligned}$$

Nyilvánvaló módon itt $S(1) = 0$ és $M(1) = 1$.

Legyen $n = 2^k$. Ekkor

$$\begin{aligned}
S(n) &= n^2 + 8S(n/2) \\
&= n^2 + 8((n/2)^2 + 8S(n/2^2)) \\
&= n^2 + 2n^2 + 8^2 S(n/2^2) \\
&= n^2 + 2n^2 + 8^2((n/2^2)^2 + 8S(n/2^3)) \\
&= n^2 + 2n^2 + 2^2 n^2 + 8^3 S(n/2^3) = \cdots \\
&= n^2 + 2n^2 + 2^2 n^2 + \cdots + 2^{k-1} n^2 + 8^k S(n/2^k) \\
&= n^2(1 + 2 + 2^2 + \cdots + 2^{k-1}) + 8^k S(n/2^k) \\
&= n^2 \, \frac{2^k - 1}{2 - 1} + 8^k S(n/2^k) \\
&= n^2(2^k - 1) + 8^k S(1) \\
&= n^2(2^k - 1) \\
&= n^2(n - 1) = n^3 - n^2.
\end{aligned}$$

$$\begin{aligned}
M(n) &= 8M(n/2) = 8^2 M(n/2^2) = 8^3 M(n/2^3) = \cdots \\
&= 8^k M(n/2^k) = 8^k M(1) = 8^k = 8^{\log_2 n} = n^{\log_2 8} = n^3.
\end{aligned}$$

Következésképpen

$$\begin{aligned}
S(n) &= \Theta(n^3), \\
M(n) &= \Theta(n^3).
\end{aligned}$$

Megmutatjuk, hogy a C_{11}, C_{12}, C_{21}, C_{22} részmátrixok kiszámításához nyolc helyett hét $n/2 \times n/2$-es mátrixszorzás is elegendő. Képezzük a következő 7 darab szorzatot:

$$\begin{aligned}
P_1 &= A_{11}(B_{12} - B_{22}), \\
P_2 &= (A_{11} + A_{12})B_{22}, \\
P_3 &= (A_{21} + A_{22})B_{11}, \\
P_4 &= A_{22}(B_{21} - B_{11}), \\
P_5 &= (A_{11} + A_{22})(B_{11} + B_{22}), \\
P_6 &= (A_{12} - A_{22})(B_{21} + B_{22}), \\
P_7 &= (A_{11} - A_{21})(B_{11} + B_{12}).
\end{aligned}$$

Ellenőrizhető, hogy ekkor

$$\begin{aligned}
C_{11} &= P_5 + P_4 - P_2 + P_6, \\
C_{12} &= P_1 + P_2, \\
C_{21} &= P_3 + P_4, \\
C_{22} &= P_5 + P_1 - P_3 - P_7.
\end{aligned}$$

A régi képletek 8 szorzást és 4 összeadást igényeltek, új módszerünkkel 7 szorzásra és 18 összeadásra (ill. kivonásra) van szükség. Így a költségre vonatkozó megfelelő rekurzív képletek most

$$\begin{aligned}
S(n) &= 7S(n/2) + 18(n/2)^2, \\
M(n) &= 7M(n/2).
\end{aligned}$$

Nyilvánvaló módon itt is $S(1) = 0$ és $M(1) = 1$.

Legyen ismét $n = 2^k$. Ekkor

$$\begin{aligned}
S(n) &= 18(n/2)^2 + 7S(n/2) \\
&= \tfrac{18}{4}n^2 + 7(18(n/2^2)^2 + 7S(n/2^2)) \\
&= \tfrac{18}{4}n^2 + \tfrac{18}{4}\,\tfrac{7}{4}n^2 + 7^2 S(n/2^2) \\
&= \tfrac{18}{4}n^2 + \tfrac{18}{4}\,\tfrac{7}{4}n^2 + 7^2(18(n/2^3)^2 + 7S(n/2^3)) \\
&= \tfrac{18}{4}n^2 + \tfrac{18}{4}\,\tfrac{7}{4}n^2 + \tfrac{18}{4}\left(\tfrac{7}{4}\right)^2 n^2 + 7^3 S(n/2^3) = \cdots \\
&= \tfrac{18}{4}n^2 + \tfrac{18}{4}\,\tfrac{7}{4}n^2 + \tfrac{18}{4}\left(\tfrac{7}{4}\right)^2 n^2 + \cdots + \tfrac{18}{4}\left(\tfrac{7}{4}\right)^{k-1} n^2 + 7^k S(n/2^k) \\
&= \tfrac{18}{4}n^2 \left(1 + \tfrac{7}{4} + \left(\tfrac{7}{4}\right)^2 + \cdots + \left(\tfrac{7}{4}\right)^{k-1}\right) + 7^k S(n/2^k) \\
&= \tfrac{18}{4}n^2\, \frac{\left(\frac{7}{4}\right)^k - 1}{\frac{7}{4} - 1} + 7^k S(n/2^k) \\
&= \tfrac{18}{4}n^2\, \frac{\left(\frac{7}{4}\right)^k - 1}{\frac{3}{4}} + 7^k S(n/2^k) \\
&= 6n^2\left(\left(\tfrac{7}{4}\right)^k - 1\right) + 7^k S(1) = 6n^2\left(\left(\tfrac{7}{4}\right)^k - 1\right) \\
&= 6n^2\left(\left(\tfrac{7}{4}\right)^{\log_2 n} - 1\right) = 6n^2(n^{\log_2 7/4} - 1) \\
&= 6n^2(n^{\log_2 7 - \log_2 4} - 1) = 6n^2(n^{(\log_2 7)-2} - 1) \\
&= 6n^{\log_2 7} - 6n^2.
\end{aligned}$$

$$\begin{aligned}
M(n) &= 7M(n/2) = 7^2 M(n/2^2) = 7^3 M(n/2^3) = \cdots \\
&= 7^k M(n/2^k) = 7^k M(1) = 7^k = 7^{\log_2 n} = n^{\log_2 7}.
\end{aligned}$$

Következésképpen

$$S(n) = \Theta(n^{\log_2 7}) = O(n^{2,81}),$$
$$M(n) = \Theta(n^{\log_2 7}) = O(n^{2,81}).$$

Ha n nem kettőhatvány, akkor legyen m a legkisebb n-et meghaladó kettőhatvány és tekintsük az

$$A^* = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} & 0 & \dots & 0 \\ a_{21} & a_{22} & \dots & a_{2n} & 0 & \dots & 0 \\ \vdots & \vdots & & \vdots & \vdots & & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nn} & 0 & \dots & 0 \\ 0 & 0 & \dots & 0 & 0 & \dots & 0 \\ \vdots & \vdots & & \vdots & \vdots & & \vdots \\ 0 & 0 & \dots & 0 & 0 & \dots & 0 \end{pmatrix}$$

és

$$B^* = \begin{pmatrix} b_{11} & b_{12} & \dots & b_{1n} & 0 & \dots & 0 \\ b_{21} & b_{22} & \dots & b_{2n} & 0 & \dots & 0 \\ \vdots & \vdots & & \vdots & \vdots & & \vdots \\ b_{n1} & b_{n2} & \dots & b_{nn} & 0 & \dots & 0 \\ 0 & 0 & \dots & 0 & 0 & \dots & 0 \\ \vdots & \vdots & & \vdots & \vdots & & \vdots \\ 0 & 0 & \dots & 0 & 0 & \dots & 0 \end{pmatrix}$$

$m \times m$-es mátrixokat. Ekkor a $C^* = A^*B^*$ szorzat egy szintén $m \times m$-es mátrix:

$$C^* = \begin{pmatrix} c_{11} & c_{12} & \dots & c_{1n} & 0 & \dots & 0 \\ c_{21} & c_{22} & \dots & c_{2n} & 0 & \dots & 0 \\ \vdots & \vdots & & \vdots & \vdots & & \vdots \\ c_{n1} & c_{n2} & \dots & c_{nn} & 0 & \dots & 0 \\ 0 & 0 & \dots & 0 & 0 & \dots & 0 \\ \vdots & \vdots & & \vdots & \vdots & & \vdots \\ 0 & 0 & \dots & 0 & 0 & \dots & 0 \end{pmatrix}.$$

A fenti algoritmust alkalmazva az A^* és B^* mátrixokra, a C^* és így a C mátrix meghatározása az előzőek szerint $O(m^{2,81}) = O((2n)^{2,81}) = O(n^{2,81})$ elemi összeadást és ugyanennyi elemi szorzást igényel.

Mátrixok szorzása még egyszer

Két $n \times n$-es mátrix összeszorzása elég gyakran jön elő különböző számítási feladatok megoldása során. Mint láttuk, a szokásos iskolai algoritmusnak ehhez $O(n^3)$ elemi aritmetikai műveletre van szüksége. A Strassen által kifejlesztett eljárást képes $O(n^{\log_2 7})$ elemei aritmetikai művelettel megoldani a feladatot. Ezt többször megjavították, a jelenlegi rekord $O(n^{2.373})$.

Itt egy kicsit egyszerűbb problémát fogunk tekinteni. Legyenek A, B, C adott $n \times n$-es mátrixok, döntsük el, hogy $AB = C$ teljesül-e. Kiszámolva az AB szorzatot ez egyszerű. Célunk egy ennél hatékonyabb algoritmus kifejlesztése. Az ötlet Freivaldstól származik. Válasszunk véletlenszerűen egy $\alpha \in \{0,1\}^n$ vektort, majd számítsuk ki a $\beta = A(B\alpha)$ és a $\gamma = C\alpha$ vektorokat. Ha $\beta = \gamma$, akkor az algoritmus térjen vissza azzal a válasszal, hogy $AB = C$, ellenkező esetben pedig azzal, hogy $AB \neq C$.

Az algoritmus csupán $O(n^2)$ elemi aritmetikai műveletet végez nyilvánvaló módon. Ha $AB = C$, akkor $\beta = \gamma$ tetszőleges $\alpha \in \{0,1\}^n$ esetén, így ebben az esetben az algoritmus biztos nem téved. Mit mondhatunk abban az esetben, ha $AB \neq C$? Megmutatjuk, hogy ekkor legalább a $\alpha \in \{0,1\}^n$ vektorok felére $\beta \neq \gamma$. Így ha α-t véletlenszerűen választjuk, akkor a tévedés valószínűsége legfeljebb $1/2$. Többször megismételve az eljárást a tévedés valószínűsége tetszőlegesen kicsivé tehető.

Tegyük fel, hogy $AB \neq C$, de $A(B\alpha) = C\alpha$, vagyis $(AB-C)\alpha$ nullvektor valamely $\alpha \in \{0,1\}^n$ esetén. Mivel $AB - C$ nem nullmátrix, ezért van olyan eleme, amelyik nullától különbözik. Legyen d egy ilyen elem, mondjuk az i-edik sor és j-edik oszlop kereszteződésében. Legyen $\alpha' \in \{0,1\}^n$ az a vektor, amelyet úgy kapunk α-ból, hogy a j-edik koordinátáját megváltoztatjuk: ha 0 volt, akkor 1 lesz, ha 1 volt, akkor 0. Ekkor $(AB - C)\alpha'$ biztos nem nullvektor, hiszen az i-edik koordinátája d-vel eltér az $(AB-C)\alpha$ nullvektor i-edik koordinátájától, így $A(B\alpha') \neq C\alpha'$. Jegyezzük még meg, hogy különböző olyan α vektorokhoz, amelyekre $A(B\alpha) = C\alpha$, különböző olyan α' vektorok tartoznak, amelyekre $A(B\alpha') \neq C\alpha'$, hiszen mindegyiknél ugyanazt a koordinátát változtatjuk meg. Innen az állítás adódik.

Polinomok szorzása

Adott egy $(m-1)$-edfokú

$$A(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_{m-1} x^{m-1}$$

és egy $(n-1)$-edfokú

$$B(x) = b_0 + b_1 x + b_2 x^2 + \cdots + b_{n-1} x^{n-1}$$

polinom. Ekkor a $C(x) = A(x)B(x)$ szorzat egy $(m+n-2)$-edfokú polinom:

$$C(x) = c_0 + c_1 x + c_2 x^2 + \cdots + c_{m+n-2} x^{m+n-2},$$

ahol

$$c_k = \sum_{\substack{i<m,\ j<n \\ i+j=k}} a_i b_j \quad (0 \leqslant k \leqslant m+n-2).$$

Ha a számítást a szokásos módon végezzük, a szorzat meghatározása $O(mn)$ elemi aritmetikai műveletet igényel. Létezik ennél hatékonyabb eljárás?

Az egyszerűség kedvéért legyen $n = m$ kettőhatvány. Értékeljük ki először az $A(x)$ és $B(x)$ polinomokat az $\omega_{j,2n} = e^{2\pi ij/2n}$ helyeken (ezek a $2n$-edik komplex egységgyökök) minden $1 \leqslant j \leqslant 2n$ esetén. Ehhez a következő oszd meg és uralkodj algoritmust használjuk.

Nem nehéz ellenőrizni, hogy egy

$$A(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_{n-1} x^{n-1}$$

polinom felírható

$$A(x) = A_{\mathrm{ps}}(x^2) + x A_{\mathrm{ptl}}(x^2).$$

alakban, ahol

$$A_{\mathrm{ps}}(x) = a_0 + a_2 x + a_4 x^2 + \cdots + a_{n-2} x^{(n-2)/2}$$

és

$$A_{\mathrm{ptl}}(x) = a_1 + a_3 x + a_5 x^2 + \cdots + a_{n-1} x^{(n-2)/2}.$$

Így

$$A(\omega_{j,2n}) = A_{\mathrm{ps}}(\omega_{j,2n}^2) + \omega_{j,2n} A_{\mathrm{ptl}}(\omega_{j,2n}^2).$$

minden $1 \leqslant j \leqslant 2n$ esetén. Vegyük még észre, hogy

$$\omega_{j,2n}^2 = (e^{2\pi ji/2n})^2 = e^{2\pi ji/n}$$

egy n-edik komplex egységgyök. Innen azonnal adódik, hogy rekurzívan kiértékelve az $(n-2)/2$-edfokú $A_{\mathrm{ps}}(x)$ és $A_{\mathrm{ptl}}(x)$ polinomokat az n-edik komplex egységgyökökön, az $A(\omega_{j,2n})$ értékek konstans számú elemi aritmetikai művelettel megkaphatók minden $1 \leqslant j \leqslant 2n$ esetén.

Jelölje $T(n)$ azon elemi aritmetikai műveletek számát, amelyek egy $(n-1)$-edfokú polinom kiértékeléséhez szükségesek a $2n$-edik komplex egyéggyökökön. A fentiek szerint

$$T(n) \leqslant 2T(n/2) + O(n).$$

Ugyanezzel a rekurzív képletettel találkoztunk az összefésüléses rendezésnél, ennek megoldása $T(n) = O(n \log n)$.

Értékeljük ki ezután a $C(x) = A(x)B(x)$ polinomot az $\omega_{j,2n}$ helyeken minden $1 \leqslant j \leqslant 2n$ esetén. Ez az $A(\omega_{j,2n})$ és $B(\omega_{j,2n})$ értékek ismeretében csupán $O(n)$ elemi aritmetikai műveletet igényel.

Nem maradt más hátra, mint a $C(x)$ polinom együtthatóinak meghatározása az $\omega_{j,2n}$ helyeken felvett értékeiből ($1 \leqslant j \leqslant 2n$). Hogyan lehet kifejezni egy legfeljebb $(2n-1)$-edfokú

$$C(x) = c_0 + c_1 x + c_2 x^2 + \cdots + c_{2n-1} x^{2n-1}$$

polinom együtthatóit a $2n$-edik egységgyökökön felvett $C(\omega_{s,2n})$ értékekből? A módszer neve gyors Fourier transzformált. Definiáljuk a

$$D(x) = d_0 + d_1 x + d_2 x^2 + \cdots + d_{2n-1} x^{2n-1}$$

polinomot, ahol $d_s = C(\omega_{s,2n})$ minden $0 \leqslant s \leqslant 2n-1$ esetén. A $D(x)$ polinomnak egy $2n$-edik egységgyökökön felvett értéke

$$\begin{aligned}
D(\omega_{j,2n}) &= \sum_{s=0}^{2n-1} C(\omega_{s,2n}) \omega_{j,2n}^s \\
&= \sum_{s=0}^{2n-1} \left(\sum_{t=0}^{2n-1} c_t \omega_{s,2n}^t \right) \omega_{j,2n}^s \\
&= \sum_{t=0}^{2n-1} c_t \left(\sum_{s=0}^{2n-1} \omega_{s,2n}^t \omega_{j,2n}^s \right) \\
&= \sum_{t=0}^{2n-1} c_t \left(\sum_{s=0}^{2n-1} e^{2\pi i (st+js)/2n} \right) \\
&= \sum_{t=0}^{2n-1} c_t \left(\sum_{s=0}^{2n-1} \omega_{t+j,2n}^s \right).
\end{aligned}$$

Vegyük azonban észre, hogy ha ω egy 1-től különböző $2n$-edik egységgyök, akkor

$$\sum_{s=0}^{2n-1} \omega^s = 0.$$

Ez azonnal következik abból a tényből, hogy ω definíció szerint gyöke az

$$x^{2n} - 1 = (x-1)(x^{2n-1} + x^{2n-2} + \cdots + x^2 + x + 1)$$

polinomnak, és mivel $\omega \neq 1$, ezért szükségképpen gyöke a jobb oldali szorzat második tényezőjének is.

Ennélfogva az utolsó sor külső összegében csak az az egy tag különbözhet nullától, amelynél $\omega_{t+j,2n} = 1$, vagyis amikor $t + j$ többszöröse $2n$-nek. Ez viszont csak akkor lehetséges, ha $t = 2n - j$. Erre az értékre

$$\sum_{s=0}^{2n-1} \omega_{t+j,2n}^{s} = \sum_{s=0}^{2n-1} 1 = 2n,$$

így

$$D(\omega_{j,2n}) = 2nc_{2n-j}.$$

A $D(\omega_{j,2n})$ értékek a már ismertetett oszd meg és uralkodj algoritmussal $O(n \log n)$ elemi aritmetikai művelettel meghatározhatók, következésképpen a $C(x)$ polinom

$$c_s = \frac{1}{2n} D(\omega_{2n-s,2n})$$

együtthatói ugyanilyen költséggel megkaphatók a $2n$-edik egységgyökökön felvett értékekből minden $0 \leqslant s \leqslant 2n - 1$ esetén.

Az algoritmus teljes költsége $O(n \log n)$.

Legközelebbi pontpár

Adott egy $\mathcal{P} = \{p_1, p_2, \ldots, p_n\}$ ponthalmaz a síkon. Szokásos módon, a $p_i = (x_i, y_i)$ és $p_j = (x_j, y_j)$ pontok távolsága

$$\sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}.$$

Adjunk hatékony algoritmust a két legközelebbi pont meghatározására (ha több ilyen pár is van, akkor megelégszünk az egyikkel)!

A legegyszerűbb megközelítési mód a következő: nézzük végig az összes pontpárt, és válasszuk ki hol lesz a távolság (négyzete) minimális. Az algoritmus költsége arányos a pontpárok számával, ami

$$\binom{n}{2} = \frac{n(n-1)}{2} = O(n^2).$$

A következőkben a legközelebbi pontpár meghatározására egy oszd meg és uralkodj algoritmust ismertetünk, amelynek költsége $O(n \log n)$.

Minden rekurzív hívásnál átadásra kerülnek a $\mathcal{P}$ ponthalmaz egy $\mathcal{Q}$ részhalmazának pontjai egy, az x koordináták szerint monoton növekvően rendezett X tömbben (azonos x koordinátájú pontok esetén a kisebb y koordinátájú jön előbb), valamint egy, az y koordináták szerint monoton növekvően rendezett Y tömbben (azonos y koordinátájú pontok esetén a kisebb x koordinátájú jön előbb). Hogy ne kelljen minden egyes rekurzív hívás előtt rendezésre pazarolni az időt, már az algoritmus elején előállítjuk $\mathcal{P}$ pontjainak az x, illetve az y koordináták szerint monoton növekvően rendezett tömbjeit, így a rekurzív hívások előtt csak kiválogatásokat kell végezni.

Egy adott rekurzív lépés a következő. Ha $|\mathcal{Q}| \leq 3$, akkor páronkénti vizsgálattal határozzuk meg a minimális távolságot. Természetesen az érdekes eset, amikor $|\mathcal{Q}| > 3$, ilyenkor az alábbiak szerint járunk el.

Először egy olyan ℓ függőleges egyenest keresünk, amely $\mathcal{Q}$-t két olyan $\mathcal{Q}_L$ és $\mathcal{Q}_R$ részre osztja, amelyekre

- $|\mathcal{Q}_L| = \lceil |\mathcal{Q}|/2 \rceil$ és $|\mathcal{Q}_R| = \lfloor |\mathcal{Q}|/2 \rfloor$,
- $\mathcal{Q}_L$ minden pontja az ℓ egyenes bal oldalán van vagy illeszkedik ℓ-re,
- $\mathcal{Q}_R$ minden pontja az ℓ egyenes jobb oldalán van vagy illeszkedik ℓ-re.

Válogassuk szét a pontokat az X tömbből az X_L és X_R tömbökbe: X_L-be kerüljenek a $\mathcal{Q}_L$-beli pontok, X_R-be pedig a $\mathcal{Q}_R$-beliek. Az X_L és X_R tömbök is rendezettek az x koordináták szerint. Válogassuk szét a pontokat az Y tömbből is az Y_L és Y_R tömbökbe: Y_L-be kerüljenek a $\mathcal{Q}_L$-beli pontok, Y_R-be pedig a $\mathcal{Q}_R$-beliek. Az Y_L és Y_R tömbök is rendezettek az y koordináták szerint.

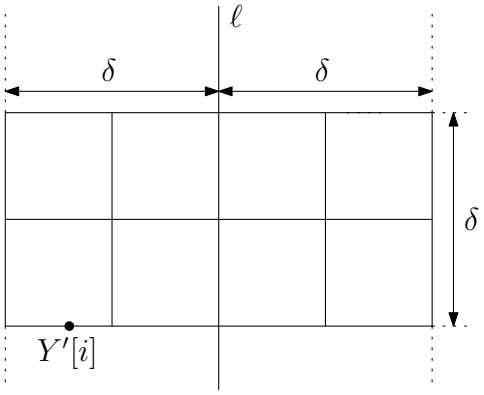
Ezután keressük meg rekurzívan a $\mathcal{Q}_L$ és $\mathcal{Q}_R$ halmazokban a legközelebbi pontpárt. Az első hívás bemenete az X_L és Y_L tömbök, míg a második hívásé az X_R és Y_R tömbök. Legyen $\mathcal{Q}_L$-ben, illetve $\mathcal{Q}_R$-ben a minimális távolság δ_L, illetve δ_R, és legyen $\delta = \min(\delta_L, \delta_R)$. Most a legközelebbi pontpár

- vagy az egyik rekurzív hívás által megtalált δ távolságú páros,
- vagy egy olyan pár, amelynek egyik pontja $\mathcal{Q}_L$-ben, a másik $\mathcal{Q}_R$-ben van.

A következő lépés annak meghatározása, hogy ez utóbbi párok között van-e olyan, amelyben a pontok távolsága kisebb, mint δ. Vegyük észre, hogy ha van ilyen pár, akkor annak egyik pontja sem lehet δ-nál nagyobb távolságra ℓ-től. Válogassuk ki az Y tömbből azokat a pontokat, amelyek

ℓ-től mért távolsága legfeljebb δ. Jelölje a kapott tömböt Y'. Az Y' tömb is rendezett az y koordináták szerint.

Most minden egyes $p \in Y'$ ponthoz keressük meg Y' azon pontjait, melyek p-től mért távolsága legfeljebb δ, és amelyek a p-n átmenő vízszintes egyenesen vagy az fölött vannak. Nem nehéz látni, hogy legfeljebb 7 ilyen pont jöhet számításba. Valóban, ha $i < j$ és $Y'[i]$ valamint $Y'[j]$ távolsága legfeljebb δ, akkor ez a két pont az ábrán látható $\delta \times 2\delta$ oldalú téglalapban van.



Ám a téglalapban lévő 8 kis négyzet egyikében sem lehet egynél több pont Y'-ből (az ℓ egyenesen lévő $\mathcal{Q}_L$-beli pontokat a bal oldali, a $\mathcal{Q}_R$-beli pontokat pedig a jobb oldali kis négyzetekbe sorolva), hiszen egy ilyen kis négyzet átmérője $\delta\sqrt{2}/2 < \delta$. Innen az állítás adódik.

Így minden $p \in Y'$ pontra elég kiszámolni p távolságát az Y'-ben utána következő 7 ponttól. Az így kapott távolságok minimuma legyen δ'. Világos, hogy ha $\delta' < \delta$, akkor $\mathcal{Q}$-ban a minimális távolság δ', egyébként a minimális távolság δ.

Jelölje $T(n)$ a költséget a kezdeti előrendezések nélkül egy n elemű ponthalmaz esetén. A rekurzív hívások költsége $T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor)$, ehhez jön még az X_L, X_R, Y_L, Y_R és Y' rendezett tömbök meghatározásának, illetve δ' kiszámításának költsége. Ez utóbbi műveletek költsége $O(n)$, így a költségre az előrendezések nélkül a

$$T(n) = \begin{cases} T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + O(n) & \text{ha } n > 3, \\ O(1) & \text{ha } n \leq 3. \end{cases}$$

rekurzív képlet adódik. Ugyanezzel a rekurzív képletettel találkoztunk az összefésüléses rendezésnél, ennek megoldása

$$T(n) = O(n \log n).$$

Mivel az előrendezések is megvalósíthatók $O(n \log n)$ költséggel (például összefésüléses rendezést használva), ezért az algoritmus teljes költsége szintén $O(n \log n)$.

Dinamikus programozás

A dinamikus programozást elsősorban optimalizálási feladatok megoldására használják. Az ilyen feladatoknak gyakran nagyon sok megoldása lehetséges. Minden megoldás egy bizonyos értékkel bír, és mi az optimális (maximális vagy minimális) értékű megoldást keressük. Sok esetben az optimális megoldás nem egyértelmű, ilyenkor általában megelégszünk egy optimális megoldás megtalálásával.

A dinamikus programozás az oszd meg és uralkodj módszerhez hasonlóan a feladatot részfeladatokra osztással oldja meg. Az oszd meg és uralkodj módszer a feladatot független részfeladatokra osztja, amelyeket rekurzívan megold, majd a részfeladatok megoldásait kombinálva kapjuk az eredeti feladat megoldását. Ezzel szemben a dinamikus programozás akkor alkalmazható, ha a részproblémák nem függetlenek, azaz közös (rész)részproblémáik vannak. Ilyen esetben az oszd meg és uralkodj módszer a szükségesnél többet dolgozik, mert ismételten megoldja a részproblémák közös (rész)részproblémáit. A dinamikus programozás minden egyes (rész)részfeladatot csak egyszer old meg és az eredményt egy táblázatban jegyzi fel, elkerülve így az ismételt számítást mikor a (rész)részfeladat megint felmerül. A dinamikus programozási algoritmusok kifejlesztése négy lépésre bontható fel:

(1) Jellemezzük az optimális megoldás szerkezetét.

(2) Rekurzív módon definiáljuk az optimális megoldás értékét.

(3) Kiszámítjuk az optimális megoldás értékét alulról felfelé történő módon.

(4) A kiszámított információk alapján megszerkesztünk egy optimális megoldást. (Ez a lépés elhagyható ha csak az optimális értékre vagyunk kíváncsiak).

Mátrixok szorzása

Tegyük fel, hogy adott n darab mátrix, jelölje ezeket $A_1, A_2, \ldots, A_n$, és az

$$A_1 A_2 \cdots A_n$$

szorzatot szeretnénk kiszámítani. Ennek a kifejezésnek az értékét megkaphatjuk a mátrixok szokásos szorzásával, amit szubrutinként használunk, miután a szorzások sorrendjét zárójelezéssel meghatározzuk. A mátrixok szorzása asszociatív, így bármilyen zárójelezés ugyanazt az eredményt adja. Például az $A_1 A_2 A_3 A_4$ szorzatot öt különböző módon lehet zárójelezni:

$$\begin{gathered}
A_1(A_2(A_3 A_4)), \\
A_1((A_2 A_3)A_4), \\
(A_1 A_2)(A_3 A_4), \\
(A_1(A_2 A_3))A_4, \\
((A_1 A_2)A_3)A_4.
\end{gathered}$$

Az a mód, ahogy egy szorzatot zárójelezünk, alapvetően befolyásolja a kifejezés kiértékelésének költségét.

Tekintsük először két mátrix összeszorzásának költségét. Az A és B mátrixot csak akkor szorozhatjuk össze, ha kompatibilisek, azaz A oszlopainak száma egyenlő B sorainak számával. Ha A egy $p \times q$ méretű, B pedig egy $q \times r$ méretű mátrix, akkor az eredményül kapott C mátrix $p \times r$ méretű. A C mátrix kiszámításához nyilván pqr darab (skalár) szorzás és ugyanennyi (szintén skalár) összeadás szükséges. A következőkben a költséget a (skalár) szorzások számával fogjuk mérni.

Annak illusztrálására, hogy a zárójelezés mennyire befolyásolja a számítás költségét, tekintsünk egy 10×100 méretű A_1, egy 100×5 méretű A_2 és egy 5×50 méretű A_3 mátrixot. Ha az $A_1 A_2 A_3$ szorzat kiszámítását az $(A_1 A_2)A_3$ zárójelezés szerint végezzük, akkor a 10×5 méretű $A_1 A_2$ mátrix kiszámításához $10 \cdot 100 \cdot 5 = 5000$ szorzást végzünk, majd ezt a mátrixot összeszorozni A_3-mal további $10 \cdot 5 \cdot 50 = 2500$ szorzás, ami összesen 7500 szorzást jelent. Ha viszont a szorzat kiszámítását az $A_1(A_2 A_3)$ zárójelezés szerint végezzük, akkor a 100×50 méretű $A_2 A_3$ mátrix kiszámításához $100 \cdot 5 \cdot 50 = 25000$ szorzást végzünk, majd ezt a mátrixot összeszorozni A_1-gyel további $10 \cdot 100 \cdot 50 = 50000$ szorzás, ami összesen 75000 szorzást jelent.

A feladat ezek után a következő: Adott mátrixoknak egy véges $A_1, A_2, \ldots, A_n$ sorozata, ahol az A_i mátrix mérete $p_{i-1} \times p_i$. Keressük meg az

$A_1A_2\cdots A_n$ szorzat azon zárójelezését, amely minimalizálja a szorzat kiszámításához szükséges (skalár) szorzások számát. Fontos megjegyezni, hogy a feladatnak nem része a szorzat kiszámítása!

Megmutatjuk, hogy az összes zárójelezés megvizsgálása nem hatékony eljárárás. Jelölje $P(n)$ egy n mátrixból álló szorzat összes zárójelezéseinek számát. Ha $n = 1$, akkor csak egyetlen mátrixunk van, amit nyilván csak egyetlen módon lehet zárójelezni. Ezután legyen $n > 1$. Ekkor a szorzat nyilván két zárójelezett szorzat szorzata. Mivel a két szorzat zárójelezéseinek száma független egymástól, így a következő rekurzív összefüggéshez jutunk:

$$P(n) = \begin{cases} 1 & \text{ha } n = 1, \\ \displaystyle\sum_{k=1}^{n-1} P(k)P(n-k) & \text{ha } n > 1. \end{cases}$$

Ennek a rekurzív egyenletnek a megoldásai a Catalan számok, amelyekről ismert, hogy $\Omega(4^n/n^{3/2})$ szerint nőnek. Tehát a zárójelezések száma exponenciális n-ben.

Az optimális zárójelezés szerkezete

Tekintsünk egy $A_iA_{i+1}\cdots A_j$ szorzatot. Ennek eredményét jelölje $A_{i..j}$. Ha a feladat nem triviális, azaz $i < j$, akkor az optimális zárójelezés az

$$A_iA_{i+1}\cdots A_j$$

szorzatot két részre vágja valamely A_k és A_{k+1} mátrixok között, ahol $i \leqslant k < j$. Az így adódó k mellett először kiszámítjuk az $A_{i..k}$ és $A_{k+1..j}$ mátrixokat, majd ezeket összeszorozva kapjuk az $A_{i..j}$ végeredményt. Ennélfogva az optimális zárójelezés költsége az $A_{i..k}$ és $A_{k+1..j}$ mátrixok kiszámításának és összeszorzásának együttes költsége.

A kulcsfontosságú észrevétel itt a következő: Az $A_iA_{i+1}\cdots A_j$ szorzat optimális zárójelezésében az $A_iA_{i+1}\cdots A_k$ és $A_{k+1}A_{k+2}\cdots A_j$ szorzatok zárójelezésének is optimálisnak kell lenni. Valóban, ha például az $A_iA_{i+1}\cdots A_k$ szorzatnak volna egy kisebb költségű zárójelezése, akkor az $A_iA_{i+1}\cdots A_j$ szorzat optimális zárójelezésében az első rész zárójelezését erre a kisebb költségűre cserélve a teljes $A_iA_{i+1}\cdots A_j$ szorzatnak is egy kisebb költségű zárójelezését kapnánk, ami ellentmondás. Erre a tulajdonságra optimális részstruktúra tulajdonságként szoktak hivatkozni.

Ebből következik, hogy az $A_iA_{i+1}\cdots A_j$ szorzat optimális zárójelezését megkaphatjuk úgy, hogy a feladatot két részre vágjuk, és az így kialakult

részfeladatok (az $A_iA_{i+1}\cdots A_k$ és $A_{k+1}A_{k+2}\cdots A_j$ szorzatok) optimális zárójelezését összetesszük. Fontos, hogy amikor keressük a kettévágás megfelelő helyét, akkor minden lehetséges pozíciót megvizsgálunk, így az optimumot adót is.

A rekurzív megoldás

A következőkben az optimális megoldás értékét rekurzívan fejezzük ki a részproblémák optimális megoldásainak értékeiből. A részproblémák az

$$A_iA_{i+1}\cdots A_j$$

alakú szorzatok optimális zárójelezésének meghatározása, ahol $1 \leqslant i \leqslant j \leqslant n$.

Jelölje $m[i,j]$ az $A_{i..j}$ mátrix kiszámításához szükséges (skalár) szorzások minimális számát. Ha $i = j$, akkor $A_{i..i} = A_i$. Ilyenkor természetesen egyetlen szorzásra sincs szükség, így $m[i,i] = 0$ minden $i = 1, 2, \ldots, n$ esetén. Ezután legyen $i < j$. Tegyük fel, hogy az optimális zárójelezés az A_k és A_{k+1} mátrixok között vágja szét az $A_iA_{i+1}\cdots A_j$ szorzatot, ahol $i \leqslant k < j$. Ekkor $m[i,j]$ megegyezik az $A_{i..k}$ és $A_{k+1..j}$ mátrixok kiszámításának minimális költségével, plusz a két mátrix összeszorzásának költségével:

$$m[i,j] = m[i,k] + m[k+1,j] + p_{i-1}p_kp_j.$$

(Ne feledjük $A_{i..k}$ egy $p_{i-1} \times p_k$ méretű, $A_{k+1..j}$ pedig egy $p_k \times p_j$ méretű mátrix.)

A fenti rekurzív egyenlet feltételezi, hogy ismerjük k értékét, bár ez nem igaz. Azonban k csak $j - i$ különböző értéket vehet fel, a szóba jövő lehetőségek $k = i, i+1, \ldots, j-1$. Mivel az optimális zárójelezés biztos használja ezek valamelyikét, ezért nincs más teendőnk, mint valamennyi esetet megvizsgálni, és a legjobbat kiválasztani. Ennélfogva

$$m[i,j] = \begin{cases} 0 & \text{ha } i = j, \\ \min\{m[i,k] + m[k+1,j] + p_{i-1}p_kp_j \mid i \leqslant k < j\} & \text{ha } i < j. \end{cases}$$

Az optimális megoldás értéke

Az optimális megoldás értékét alulról felfelé történő módon határozzuk meg. Az algoritmus bemenete a $p = (p_0, p_1, \ldots, p_n)$ sorozat.

```
MátrixSzorzás(p)
for i=1 to n do
   m[i,i]=0
for l=2 to n do
   for i=1 to n-l+1 do
      j=i+l-1
      m[i,j]=INFINITY
      for k=i to j-1 do
         q=m[i,k]+m[k+1,j]+p_{i-1}p_kp_j
         if q<m[i,j] then m[i,j]=q
return m
```

Az algoritmus az első két sorban az $m[i,i] = 0$ értékadásokat hajtja végre. Ezután az $m[i,i+1]$ értékeket, azaz az $l = 2$ hosszú szorzatok minimális költségét számítja ki a harmadik sorban kezdődő ciklus első lefutásakor. A ciklus második lefutásakor következnek az $m[i,i+2]$ értékek, azaz az $l = 3$ hosszú szorzatok minimális költségei, és így tovább. Jegyezzük meg, hogy amikor az $m[i,j]$ mennyiséget határozzuk meg, akkor az ehhez szükséges $m[i,k]$ és $m[k+1,j]$ mennyiségek rendelkezésünkre állnak, azokat már előbb kiszámítottuk. Mivel az $m[i,j]$ mennyiségek csak $i \leqslant j$ esetén definiáltak, ezért az m tömb elemei közül csak a főátló fölé eső ket használjuk.

A `MátrixSzorzás(p)` eljárás három egymásba ágyazott ciklusának egyszerű vizsgálata $O(n^3)$ lépés, mivel a három ciklusváltozó mindegyike (l, i és k) legfeljebb n értéket vehet fel.

Egy optimális megoldás

Az optimális megoldás megtalálása érdekében még egy jelölést vezetünk be. Jelölje $s[i,j]$ azt a k indexet, ahol az $A_iA_{i+1}\cdots A_j$ szorzatot az optimális zárójelezés kettévágja, vagyis azt a k indexet, amelyre

$$m[i,j] = m[i,k] + m[k+1,j] + p_{i-1}p_kp_j.$$

Az s tömb a `MátrixSzorzás(p)` eljárás során könnyen számítható.

```
MátrixSzorzás(p)
for i=1 to n do
   m[i,i]=0
for l=2 to n do
   for i=1 to n-l+1 do
      j=i+l-1
```

```
      m[i,j]=INFINITY
      for k=i to j-1 do
         q=m[i,k]+m[k+1,j]+p_{i-1}p_kp_j
         if q<m[i,j] then
            m[i,j]=q
            s[i,j]=k
return m,s
```

Az s tömb felhasználásával már könnyű a mátrixok összeszorzásának leggazdaságosabb módját megtalálni. Az $s[1,n]$ elem azt a k indexet tartalmazza, ahol az optimális zárójelezés az $A_1A_2\cdots A_n$ szorzatot kettévágja A_k és A_{k+1} között. Tehát a teljes $A_{1..n}$ szorzat optimális kiszámításakor a végső mátrixszorzás

$$A_{1..s[1,n]}A_{s[1,n]+1..n}.$$

A korábbi mátrixszorzásokat rekurzívan számíthatjuk ki:

- $s[1,s[1,n]]$ adja meg az utolsó szorzás helyét $A_{1..s[1,n]}$ számításakor,
- $s[s[1,n]+1,n]$ adja meg az utolsó szorzás helyét $A_{s[1,n]+1..n}$ számításakor.

Az eljárást az $(s,1,n)$ paraméterekkel kell meghívni.

```
OptimálisZárójelezés(s,i,j)
if j=i
   then
      print 'A'_i
   else
      print '('
      OptimálisZárójelezés(s,i,s[i,j])
      OptimálisZárójelezés(s,s[i,j]+1,j)
      print ')'
```

Szekvenciák hasonlósága

Biológiai alkalmazásokban gyakran akarjuk összehasonlítani két élőlény DNS láncát. A DNS lánc bizonyos – bázisnak nevezett – molekulákból áll. A lehetséges bázisok adenin, guanin, citozin és timin. A bázisokat a kezdőbetűjűkkel jelölve, a DNS lánc leírható egy a véges $\{A,C,G,T\}$ halmaz feletti sztringgel. Például az egyik élőlény esetén egy szekvencia lehet

$$S_1 = ACCGGTCGAGTGCGCGGAAGCCGGCCGAA,$$

míg a másiknál

$$S_2 = GTCGTTCGGAATGCCGTTGCTCTGTAAA.$$

Minél hasonlóbb a két DNS lánc (szekvencia), az élőlények annál "közelebb" állnak egymáshoz. A hasonlóság persze sokféleképpen mérhető:

- Például azt mondhatjuk, hogy a két DNS lánc akkor hasonló, ha egyik a másiknak részsztringje.
- Azt is mondhatjuk, hogy két lánc akkor hasonló, ha kevés változtatással átvihetők egymásba.
- Egy harmadik lehetőség, hogy keresünk egy olyan S_3 szekvenciát, amelynek a bázisai megjelennek S_1-ben és S_2-ben is, ráadásul ugyanabban a sorrendben, de nem feltétlenül egymás után. Minél hosszabb ilyen S_3 található, annál nagyobb a hasonlóság. Példánkban

$$S_3 = GTCGTCGGAAGCCGGCCGAA.$$

Leghosszabb közös részsorozat

Egy $Z = (z_1, z_2, \ldots, z_k)$ sorozat részsorozata egy $X = (x_1, x_2, \ldots, x_n)$ sorozatnak, ha léteznek olyan $1 \leqslant i_1 < i_2 < \cdots < i_k \leqslant n$ indexek, hogy minden $1 \leqslant j \leqslant k$ esetén $z_j = x_{i_j}$. Azt mondjuk, hogy egy Z sorozat közös részsorozata az X és Y sorozatoknak, ha Z részsorozata X-nek és Y-nak is. Adjunk hatékony algoritmust az $X = (x_1, x_2, \ldots, x_n)$ és $Y = (y_1, y_2, \ldots, y_m)$ sorozatok egy leghosszabb közös részsorozatának meghatározására!

A feladatot dinamikus programozással oldjuk meg. Szükségünk lesz a prefix fogalmára. Az $X = (x_1, x_2, \ldots, x_n)$ sorozat $X_i = (x_1, x_2, \ldots, x_i)$ részsorozatát az X sorozat i-edik prefixének nevezzük tetszőleges $0 \leqslant i \leqslant n$ esetén.

Állítás. Legyen $X = (x_1, x_2, \ldots, x_n)$ és $Y = (y_1, y_2, \ldots, y_m)$ két sorozat és $Z = (z_1, z_2, \ldots, z_k)$ ezek egy leghosszabb közös részsorozata. Ekkor

(1) ha $x_n = y_m$, akkor $z_k = x_n = y_m$ és Z_{k-1} egy leghosszabb közös részsorozata X_{n-1}-nek és Y_{m-1}-nek,

(2) ha $x_n \neq y_m$ és $z_k \neq x_n$, akkor Z egy leghosszabb közös részsorozata X_{n-1}-nek és Y-nak,

(3) ha $x_n \neq y_m$ és $z_k \neq y_m$, akkor Z egy leghosszabb közös részsorozata X-nek és Y_{m-1}-nek.

Bizonyítás.
(1) Ha $z_k \neq x_n$, akkor az $x_n = y_m$ elemet hozzávehetnénk Z-hez, ami által X és Y egy $k+1$ hosszú közös részsorozatát kapnánk, ellentmondás. Így $z_k = x_n = y_m$. Világos, hogy Z_{k-1} közös részsorozata X_{n-1}-nek és Y_{m-1}-nek. Ha lenne ennél hosszabb közös részsorozata X_{n-1}-nek és Y_{m-1}-nek, akkor ehhez z_k-t hozzáfűzve X-nek és Y-nak egy Z-nél hosszabb közös részsorozatát kapnánk, ami nem lehetséges. Így Z_{k-1} szükségképpen egy leghosszabb közös részsorozata X_{n-1}-nek és Y_{m-1}-nek.

(2) Ha $z_k \neq x_n$, akkor Z közös részsorozata X_{n-1}-nek és Y-nak. Ha X_{n-1}-nek és Y-nak volna Z-nél hosszabb közös részsorozata, az Z-nél hosszabb közös részsorozata lenne X-nek és Y-nak is, ellentmondás.

(3) Ugyanúgy, mint az előbb.

Az előző állítás szerint csak egy vagy két részfeladat van, amit meg kell vizsgálni az $X = (x_1, x_2, \ldots, x_n)$ és $Y = (y_1, y_2, \ldots, y_m)$ sorozatok egy leghosszabb közös részsorozatának megkeresésekor.

- Ha $x_n = y_m$, akkor elég X_{n-1} és Y_{m-1} egy leghosszabb közös részsorozatát megtalálni, amelyhez az $x_n = y_m$ elemet csatolva X és Y egy leghosszabb közös részsorozatához jutunk.
- Ha $x_n \neq y_m$, akkor két részfeladatot kell megoldani: X_{n-1} és Y, illetve X és Y_{m-1} egy-egy leghosszabb közös részsorozatát kell megkeresni. Ezek közül a hosszabb X-nek és Y-nak is egy leghosszabb közös részsorozata lesz.

Ezek után az X_i és Y_j sorozatok leghosszabb közös részsorozatának hosszát jelölje $c[i,j]$ minden $0 \leqslant i \leqslant n$ és $0 \leqslant j \leqslant m$ esetén. Ha $i = 0$ vagy $j = 0$, azaz a sorozatok legalább egyikének a hossza nulla, akkor természetesen a leghosszabb közös részsorozat hossza is nulla. A többi esetben a fenti észrevétel igazít el. Mindent összevetve a $c[i,j]$-re vonatkozó rekurzív képlet a következő:

$$c[i,j] = \begin{cases} 0 & \text{ha } i = 0 \text{ vagy } j = 0, \\ c[i-1, j-1] + 1 & \text{ha } i, j > 0 \text{ és } x_i = y_j, \\ \max(c[i-1, j], c[i, j-1]) & \text{ha } i, j > 0 \text{ és } x_i \neq y_j. \end{cases}$$

Egy leghosszabb közös részsorozat megtalálása érdekében még egy jelölést vezetünk be. Minden $1 \leqslant i \leqslant n$ és $1 \leqslant j \leqslant m$ esetén jelölje $b[i,j]$ a c tömb

azon elemének indexét, ahonnan $c[i,j]$ értékének meghatározásakor a legjobb értéket kaptuk.

```
LeghosszabbKözösRészsorozat(X,n,Y,m)
for i=1 to n do
   c[i,0]=0
for j=0 to m do
   c[0,j]=0
for i=1 to n do
   for j=1 to m do
      if x_i=y_j
         then
            c[i,j]=c[i-1,j-1]+1
            b[i,j]=(i-1,j-1)
         else
            if c[i-1,j]>=c[i,j-1]
               then
                  c[i,j]=c[i-1,j]
                  b[i,j]=(i-1,j)
               else
                  c[i,j]=c[i,j-1]
                  b[i,j]=(i,j-1)
return c,b
```

A $c[0:n,0:m]$ tömb elemeit sorfolytonosan töltjük ki, először az első sort balról jobbra, aztán a másodikat, és így tovább. A leghosszabb közös részsorozat hossza $c[n,m]$. Az eljárás költsége nyilván $O(nm)$.

A b tömb felhasználásával könnyű egy leghosszabb közös részsorozatot megtalálni: a $b[n,m]$ elemből indulva egyszerűen csak végig kell haladni a "mutatók" mentén a tömbön. Amikor egy $b[i,j]$ elemről a $b[i-1,j-1]$ elemre lépünk, az azt jelenti, hogy az $x_i = y_j$ elem benne van a leghosszabb közös részsorozatban. Ezzel a módszerrel a leghosszabb közös részsorozat elemeit fordított sorrendben kapjuk meg.

A következő rekurzív eljárás a leghosszabb közös részsorozat elemeit a helyes, eredeti sorrendben nyomtatja ki. Az eljárást a (b,X,n,m) paraméterekkel kell meghívni.

```
Nyomtat(b,X,i,j)
if i=0 OR j=0 then return
if b[i,j]=(i-1,j-1)
```

```
then
   Nyomtat(b,X,i-1,j-1)
   print x_i
else
   if b[i,j]=(i-1,j)
      then Nyomtat(b,X,i-1,j)
      else Nyomtat(b,X,i,j-1)
```

Az eljárás költsége $O(n+m)$, hiszen minden rekurzív hívásnál i és j legalább egyikének csökken az értéke.

Szekvenciaillesztés

Tegyük fel, hogy adottak az $X = (x_1, x_2, \ldots, x_m)$ és $Y = (y_1, y_2, \ldots, y_n)$ karaktersorozatok. Tekintsük az $\{1, 2, \ldots, m\}$ és $\{1, 2, \ldots, n\}$ halmazokat, amelyekkel az X és Y karaktersorozatok pozícióit reprezentáljuk, és tekintsük a két halmaz egy M párosítását. Egy ilyen M párosítást szekvenciaillesztésnek nevezünk, ha tetszőleges $(i, j), (i', j') \in M$ és $i < i'$ esetén $j < j'$.

A szekvenciaillesztéseknek létezik egy elég természetes ábrázolási módja. Tekintsük például a `gatcggcat` és `caatgtgaatc` szekvenciákat és az

$$\{(1,1), (2,3), (3,4), (5,5), (6,7), (7,8), (8,9), (9,10)\}$$

szekvenciaillesztést. Ezt a következőképpen szemléltethetjük:

```
g - a t c g - g c a t -
c a a t - g t g a a t c
```

Úgy is interpretálhatjuk ezt a képet, hogy a `gatcggcat` szekvenciától eljuthatunk a `caatgtgaatc` szekvenciához, ha az első pozíción levő `g` karaktert felcseréljük a `c` karakterre, majd az első pozíció után beszúrjuk az `a` karaktert, ezután töröljük a negyedik pozíción levő `c` karaktert, majd az ötödik pozíció után beszúrjuk a `t` karaktert, aztán a hetedik pozíción levő `c` karaktert felcseréljük az `a` karakterre, végül a kilencedik pozíció után beszúrjuk a `c` karaktert.

Legyen M egy szekvenciaillesztés az X és Y karaktersorozatok között. Az M szekvenciaillesztés költségét a következőképpen definiáljuk.

- Létezik egy $g > 0$ konstans, amely a hézag költségét reprezentálja. Az X és Y minden olyan pozíciójára, amely nem szerepel egyik M-beli párban sem, felszámolunk g költséget.

- Az ábécé minden ("p","q") karakterpárjához létezik egy c["p", "q"] $\geqslant 0$ konstans, amely a "p" karakternek a "q" karakterhez párosításának költségét reprezentálja (általában feltesszük, hogy c["p", "p"] $= 0$ az ábécé tetszőleges "p" karakterére, bár ez nem szükségszerű). Minden $(i, j) \in M$ párra felszámolunk $c[x_i, y_j]$ költséget.

Az M szekvenciaillesztés költsége legyen ezek után a fent definiált költségek összege.

Adjunk hatékony algoritmust az X és Y karaktersorozatok minimális költségű szekvenciaillesztésének meghatározására! Minél kisebb ez a költség, annál hasonlóbbnak fogjuk tekinteni az X és Y karaktersorozatokat.

A feladatot dinamikus programozással oldjuk meg. Szükségünk lesz egy egyszerű észrevételre.

Állítás. Legyen M optimális szekvenciaillesztés az X és Y karaktersorozatok között. Ekkor a következők legalább egyike fennáll:

(1) $(m, n) \in M$,

(2) az X karaktersorozat m-edik pozíciója nem szerepel az M-beli párok egyikében sem,

(3) az Y karaktersorozat n-edik pozíciója nem szerepel az M-beli párok egyikében sem.

Bizonyítás. Indirekt tegyük fel, hogy $(m, n) \notin M$ és léteznek olyan $i < m$ és $j < n$ pozitív egészek, amelyekre $(m, j) \in M$ és $(i, n) \in M$. Azonban ez ellentmond a szekvenciaillesztés definíciójának: itt $(i, n), (m, j) \in M$ és $i < m$, miközben $n > j$. Innen az állítás adódik.

Állítás. Legyen M optimális szekvenciaillesztés az X és Y karaktersorozatok között.

(1) Ha $(m, n) \in M$, akkor $M \setminus \{(m, n)\}$ optimális szekvenciaillesztés az X_{m-1} és Y_{n-1} prefixek között.

(2) Ha az X karaktersorozat m-edik pozíciója nem szerepel az M-beli párok egyikében sem, akkor M optimális szekvenciaillesztés az X_{m-1} prefix és Y között.

(3) Ha az Y karaktersorozat n-edik pozíciója nem szerepel az M-beli párok egyikében sem, akkor M optimális szekvenciaillesztés X és az Y_{n-1} prefix között.

Bizonyítás.
(1) Ha lenne $M \setminus \{(m,n)\}$-nél kisebb költségű szekvenciaillesztés az X_{m-1} és Y_{n-1} prefixek között, akkor ehhez hozzávéve az (m,n) párt egy M-nél kisebb költségű szekvenciaillesztést kapnánk X és Y között, ami ellentmond M optimalitásának.

(2) Ha lenne M-nél kisebb költségű szekvenciaillesztés az X_{m-1} prefix és Y között, akkor ugyanez M-nél kisebb költségű szekvenciaillesztés lenne X és Y között is, ami ellentmond M optimalitásának.

(3) Ha lenne M-nél kisebb költségű szekvenciaillesztés X és az Y_{n-1} prefix között, akkor ugyanez M-nél kisebb költségű szekvenciaillesztés lenne X és Y között is, ami ellentmond M optimalitásának.

Mit jelent ez?

- Ha az X és Y karaktersorozatok közötti optimális szekvenciaillesztés tartalmazza az (m,n) párt, akkor az optimális szekvenciaillesztés költsége nyilván $c[x_m, y_n]$ plusz az X_{m-1} és Y_{n-1} prefixek közötti optimális szekvenciaillesztés költsége.

- Ha az X és Y karaktersorozatok közötti optimális szekvenciaillesztés egyik párjában sem szerepel az X karaktersorozat m-edik pozíciója, akkor az optimális szekvenciaillesztés költsége nyilván g plusz az X_{m-1} prefix és Y közötti optimális szekvenciaillesztés költsége.

- Ha az X és Y karaktersorozatok közötti optimális szekvenciaillesztés egyik párjában sem szerepel az Y karaktersorozat n-edik pozíciója, akkor az optimális szekvenciaillesztés költsége nyilván g plusz az X és az Y_{n-1} prefix közötti optimális szekvenciaillesztés költsége.

Az optimális szekvenciaillesztés költségének meghatározásánál mindhárom lehetőséget számba kell venni és a legkisebb költségűt választani nyilvánvaló módon.

Jelölje ezek után $a[i,j]$ az X_i és Y_j prefixek közötti optimális szekvenciaillesztés költségét minden $0 \leqslant i \leqslant m$ és $0 \leqslant j \leqslant n$ esetén. Nyilván $a[i,0] = ig$ és $a[0,j] = jg$ minden $0 \leqslant i \leqslant m$ és $0 \leqslant j \leqslant n$ esetén. A többi esetben pedig az előző észrevétellel összhangban

$$a[i,j] = \min(c[x_i, y_j] + a[i-1, j-1], g + a[i-1, j], g + a[i, j-1]).$$

Egy minimális költségű szekvenciaillesztés megtalálása érdekében még egy jelölést vezetünk be. Minden $1 \leqslant i \leqslant n$ és $1 \leqslant j \leqslant m$ esetén jelölje $b[i,j]$ az a tömb azon elemének indexét, ahonnan $a[i,j]$ értékének meghatározásakor a legkisebb értéket kaptuk.

```
Szekvenciaillessztés(X,m,Y,n)
for i=1 to m do
  a[i,0]=i*g
for j=0 to n do
  a[0,j]=j*g
for i=1 to m do
  for j=1 to n do
    a[i,j]=c[x_i,y_j]+a[i-1,j-1]
    b[i,j]=(i-1,j-1)
    if a[i,j]>g+a[i-1,j] then
      a[i,j]=g+a[i-1,j]
      b[i,j]=(i-1,j)
    if a[i,j]>g+a[i,j-1] then
      a[i,j]=g+a[i,j-1]
      b[i,j]=(i,j-1)
return a,b
```

Az $a[0:m,0:n]$ tömb elemeit sorfolytonosan töltjük ki, először az első sort balról jobbra, aztán a másodikat, és így tovább. Az optimális szekvenciaillesztés költsége $a[m,n]$. Az eljárás költsége nyilván $O(mn)$.

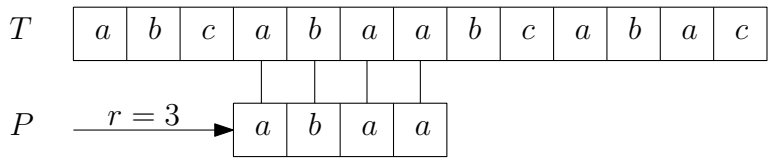
A b tömb felhasználásával könnyű egy optimális szekvenciaillesztést megtalálni: a $b[m,n]$ elemből indulva egyszerűen csak végig kell haladni a "mutatók" mentén a tömbön. Amikor egy $b[i,j]$ elemről a $b[i-1,j-1]$ elemre lépünk, az azt jelenti, hogy az (i,j) pár hozzátartozik az optimális szekvenciaillesztéshez. A következő rekurzív eljárás az optimális szekvenciaillesztés elemeit nyomtatja ki. Az eljárást a (b,m,n) paraméterekkel kell meghívni.

```
Nyomtat(b,i,j)
if i=0 OR j=0 then return
if b[i,j]=(i-1,j-1)
  then
    Nyomtat(b,i-1,j-1)
    print '(' i ',' j ')'
  else
    if b[i,j]=(i-1,j)
      then Nyomtat(b,i-1,j)
      else Nyomtat(b,i,j-1)
```

Az eljárás költsége $O(m+n)$, hiszen minden rekurzív hívásnál i és j legalább egyikének csökken az értéke.

Mintaillesztés

Szövegszerkesztő programokban is gyakori feladat megkeresni egy szövegben egy minta összes előfordulását. Tegyük fel, hogy a szöveget egy n elemű $T[1:n]$ tömb, a mintát pedig egy m elemű $P[1:m]$ tömb tartalmazza, ahol $m \leqslant n$. Azt mondjuk, hogy a P minta r eltolással előfordul a T szövegben, vagy másképpen fogalmazva a T szöveg $(r+1)$-edik pozíciójára illeszkedik, ha $0 \leqslant r \leqslant n-m$ és $T[r+1:r+m] = P[1:m]$, azaz minden $j = 1, 2, \ldots, m$ esetén $T[r+j] = P[j]$. Ha a P minta r eltolással előfordul T-ben, akkor r érvényes eltolás, ellenkező esetben r érvénytelen eltolás.



Adjunk hatékony algoritmust egy adott P minta összes érvényes eltolásának megtalálására egy rögzített T szövegben!

Kezdjük néhány elnevezéssel. Az x és y karakterláncok xy konkatenációja az a karakterlánc, melyben x karaktereit y karakterei követik. A w karakterlánc az x karakterlánc valódi prefixe, ha van olyan nem üres y karakterlánc, hogy $x = wy$. Hasonlóan, a w karakterlánc az x karakterlánc valódi szuffixe, ha van olyan nem üres y karakterlánc, hogy $x = yw$. A w karakterlánc az x karakterlánc szegélye, ha w az x leghosszabb olyan valódi prefixe, amely egyben valódi szuffixe is x-nek. Tehát w a leghosszabb olyan karakterlánc, amelyre alkalmas nem üres u és v karakterláncokkal $x = wu$ és $x = vw$.

Először a P mintához előállítunk egy $S[1:m]$ tömböt, ahol $S[j]$ definíció szerint a $P[1:j]$ karakterlánc szegélyének hossza minden $1 \leqslant j \leqslant m$ esetén. Ehhez dinamikus programozást használunk.

Nyilvánvaló módon $S[1] = 0$. Tegyük fel ezután, hogy $j > 1$ és az $S[1:j-1]$ résztömböt már kitöltöttük. Ekkor $S[j]$ a következőképpen számítható.

Jelölje w a $P[1:j]$ karakterlánc szegélyét és l a w hosszát. Ekkor $S[j] = l$. Először is jegyezzük meg, hogy $S[j]$ értéke legfeljebb $S[j-1]+1$ lehet, hiszen ha $P[1:l] = P[j-l+1:j]$, akkor $P[1:l-1] = P[j-l+1:j-1]$ is nyilván igaz, vagyis a $P[1:j-1]$ karakterláncnak van olyan $l-1$ hosszú valódi prefixe, amely egyben valódi szuffixe is.

Ha $P[j] = P[S[j-1]+1]$, akkor $S[j] = S[j-1]+1$, és viszont. Ezután tegyük fel, hogy $P[j] \neq P[S[j-1]+1]$. Ekkor persze $S[j] \leqslant S[j-1]$. Most a $w[1:l-1]$ karakterlánc valódi prefixe a $P[1:S[j-1]]$ karakterláncnak

és valódi szuffixe a $P[j - S[j-1] : j-1]$ karakterláncnak. Ám ez utóbbi két karakterlánc az S tömb definíciója szerint azonos, így a w karakterlánc valódi prefixe és egyben valódi szuffixe is annak a karakterláncnak, amit úgy kapunk, hogy a $P[1 : S[j-1]]$ karakterlánc végére fűzzük a $P[j]$ karaktert. Vegyük észre, hogy ha lenne ennek a karakterláncnak l-nél hosszabb olyan valódi prefixe, amely egyben valódi szuffixe is, akkor ez a $P[1 : j]$ karakterláncnak szintén egy l-nél hosszabb olyan valódi prefixe lenne, amely egyben valódi szuffixe is, ellentmondás. Következésképpen w szegélye annak a karakterláncnak, amit úgy kapunk, hogy a $P[1 : S[j-1]]$ karakterlánc végére fűzzük a $P[j]$ karaktert.

Lássuk ezek után az S tömb elemeit számító eljárást.

```
SzegélySzámítás(P,m)
S[1]=0
k=0
for j=2 to m do
  while k>0 AND P[k+1]<>P[j] do
    k=S[k]
  if P[k+1]=P[j] then k=k+1
  S[j]=k
return S
```

Meghatározzuk az S tömb kitöltésének a karakterösszehasonlítások számában mért költségét. Jelölje b_j az $S[j]$ elem számításánál felhasznált karakterösszehasonlítások számát (ebbe most $S[1 : j-1]$ számításának költségét nem értjük bele). A k változó értéke kezdetben $S[j-1]$, ez a while ciklus magjának minden lefutása után legalább eggyel csökken. A ciklusmag legalább $b_j - 2$ alkalommal végrehajtásra kerül, ezért a while ciklusból kilépve k értéke legfeljebb $S[j-1] - b_j + 2$. Ezután k értéke eggyel nőhet még, így végső értéke, ami $S[j]$, legfeljebb $S[j-1] - b_j + 3$. Ennélfogva

$$b_j \leqslant S[j-1] - S[j] + 3.$$

Az összehasonlítások száma ezek után

$$\sum_{j=2}^{m} b_j = \sum_{j=2}^{m} (S[j-1] - S[j] + 3) = S[1] - S[m] + 3(m-1) \leqslant 3(m-1).$$

Így az S tömb kitöltésének költége $O(m)$.

Ezután lássuk magát az algoritmust. Tegyük fel, hogy éppen a T szöveg i-edik karakterét a P minta $(k+1)$-edik karakterével hasonlítjuk össze, és

már tudjuk, hogy $T[i-k] = P[1], \ldots, T[i-1] = P[k]$, azaz a minta első k karakterénél illeszkedést találtunk.

Először tegyük fel, hogy $P[k+1] = T[i]$. Ha $k+1 < m$, akkor a szövegben és a mintában is egy hellyel jobbra lépünk (k és i értéke eggyel nő). Ha $k+1 = m$, akkor találtunk egy P-vel azonos részkarakterláncot T-ben. Ilyenkor a szövegben egy hellyel jobbra lépünk, a mintában pedig $m - S[m]$ hellyel vissza (k értéke $S[m]$ lesz, i értéke eggyel nő). Most egyrészt P első $S[m]$ darab karaktere illeszkedni fog a T szöveg i-edik karaktere előtti $S[m]$ hosszú részhez, másrészt ha kevesebb hellyel lépünk vissza, akkor S definíciója miatt a T szöveg i-edik karaktere előtt nem illeszkedhet minden karakter.

Tegyük fel ezután, hogy $P[k+1] \neq T[i]$. Ha $k = 0$, azaz éppen P első karakterénél tartunk, akkor a T szövegben egy hellyel jobbra lépünk (k értéke nem változik, i értéke eggyel nő). Ha $k > 0$, akkor a mintában $k - S[k]$ hellyel visszalépünk (k értéke $S[k]$ lesz, i nem változik). Ez utóbbi esetben egyrészt P első $S[k]$ darab karaktere illeszkedni fog a T szöveg i-edik karaktere előtt lévő $S[k]$ hosszú részhez, másrészt ha kevesebb hellyel lépünk vissza, akkor S definíciója miatt a T szöveg i-edik karaktere előtt nem illeszkedhet minden karakter.

```
Mintaillesztés(T,n,P,m)
S=SzegélySzámítás(P,m)
k=0
for i=1 to n do
  while k>0 AND P[k+1]<>T[i] do
    k=S[k]
  if P[k+1]=T[i] then k=k+1
  if k=m then
    print 'A minta illeszkedik a(z) ' i-m+1 '. pozícióra'
    k=S[k]
```

A költséget itt is a karakterösszehasonlítások számában mérjük. Vizsgáljuk az i és az $i-k$ mennyiségeket az egymás utáni karakterösszehasonlítások pillanataiban. Az első karakterösszehasonlításkor $i = 1$ és $k = 0$, tehát $i - k = 1$. Vegyük észre, hogy az i és $i - k$ mennyiségek semelyik karakterösszehasonlításkor sem kisebbek, mint az azt megelőző karakterösszehasonlításkor. Vegyük észre azt is, hogy a while ciklusok utáni n darab karakterösszehasonlítást leszámítva, az i és $i-k$ mennyiségek legalább egyike minden karakterösszehasonlításkor nagyobb, mint az azt megelőző karakterösszehasonlításkor. Innen $i \leqslant n$ és $i - k \leqslant n$ felhasználásával következik, hogy a karakterösszehasonlítások száma összesen legfeljebb $2n - 1 + n = 3n - 1$.

Így az algoritmus teljes költsége az S tömb számításával együtt $O(m+n)$.

Optimális bináris keresőfa

Adott különböző kulcsoknak egy $k_1 < k_2 < \cdots < k_n$ rendezett sorozata, ezekből szeretnénk egy bináris keresőfát felépíteni. Ha minden k_i kulcsnak ismert a p_i keresési valószínűsége, akkor egy adott T bináris keresőfára megállapítható a keresés költségének várható értéke. Egy kulcs megtalálásának a költsége legyen a keresés során megvizsgált kulcsok száma, azaz a kulcs mélysége a fában (a gyökértől a kulcsot tartalmazó csúcsig vezető út hossza) plusz egy. Így a keresés költségének várható értéke

$$\sum_{i=1}^{n}(d_T(k_i)+1)p_i.$$

A kulcsok keresési valószínűségeinek ismeretében olyan bináris kersőfát szeretnénk szerkeszteni, amelyre a keresés költségének várható értéke minimális. Egy ilyen fát optimális bináris keresőfának nevezünk.

A feladatot dinamikus programozással oldjuk meg. Először is jegyezzük meg, hogy egy a $k_1, k_2, \ldots, k_n$ kulcsokat tartalmazó bináris keresőfa bármely részfája pontosan a $k_i, k_{i+1}, \ldots, k_j$ kulcsokat tartalmazza valamely $1 \leqslant i \leqslant j \leqslant n$ esetén (gondoljunk a fa inorder bejárására).

Minden $1 \leqslant i \leqslant j \leqslant n$ esetén jelölje $c[i,j]$ azon feladat egy optimális megoldásában a

$$\sum_{s=i}^{j}(d_T(k_s)+1)p_s.$$

értéket, amikor a $k_i, k_{i+1}, \ldots, k_j$ kulcsokból építünk bináris keresőfát. A formulák kompakt felírhatóságának érdekében legyen $c[i,i-1] = 0$, minden $1 \leqslant i \leqslant n+1$ esetén.

Legyen most $1 \leqslant i \leqslant j \leqslant n$, továbbá legyen $\mathcal{T}$ egy optimális megoldása annak a feladatnak, amikor a $k_i, k_{i+1}, \ldots, k_j$ kulcsokból építünk bináris keresőfát. Tegyük fel, hogy a $\mathcal{T}$ gyökerében a k_h kulcs van. A szokásos módon igazolható, hogy ekkor $\mathcal{T}$ bal oldali $\mathcal{T}'$ részfája egy optimális megoldása annak a feladatnak, amikor a $k_i, \ldots, k_{h-1}$ kulcsokból építünk bináris keresőfát, míg $\mathcal{T}$ jobb oldali $\mathcal{T}''$ részfája egy optimális megoldása annak a feladatnak, amikor a $k_{h+1}, \ldots, k_j$ kulcsokból építünk bináris keresőfát (optimális részstruktúra tulajdonság). Ennélfogva

$$\begin{aligned} c[i,j] &= c[i,h-1] + \sum_{s=i}^{h-1} p_s + p_h + c[h+1,j] + \sum_{s=h+1}^{j} p_s \\ &= c[i,h-1] + c[h+1,j] + \sum_{s=i}^{j} p_s, \end{aligned}$$

hiszen $\mathcal{T}'$ és $\mathcal{T}''$ minden csúcsának mélysége eggyel nagyobb lesz $\mathcal{T}$-ben. Ez a rekurzív egyenlet feltételezi, hogy ismerjük h értékét, bár ez nem igaz. Azonban h csak $j-i+1$ különböző értéket vehet fel, a szóba jövő lehetőségek $h = i, i+1, \ldots, j$. Mivel az optimális megoldás biztos használja ezek valamelyikét, ezért nincs más teendőnk, mint valamennyi esetet megvizsgálni, és a legjobbat kiválasztani. Ennélfogva

$$c[i,j] = \min_{i \leqslant h \leqslant j} \left\{ c[i,h-1] + c[h+1,j] + \sum_{s=i}^{j} p_s \right\}.$$

A $c[i,i-1] = 0$ értékadások végrehajtása után először a $c[i,i]$, ezután a $c[i,i+1]$ értékeket számítjuk ki, majd következnek a $c[i,i+2]$ értékek, és így tovább (a főátlóval párhuzamos átlók mentén). Jegyezzük meg, hogy amikor a $c[i,j]$ értéket határozzuk meg, akkor az ehhez szükséges $c[i,h-1]$ és $c[h+1,j]$ értékek rendelkezésünkre állnak, azokat már előbb kiszámítottuk. Az algoritmus költsége $O(n^3)$.

Egy optimális bináris keresőfa megszerkesztéséhez bevezetünk egy $d[i,j]$ tömböt, amely megadja egy a $k_i, k_{i+1}, \ldots, k_j$ kulcsokból álló optimális bináris keresőfa gyökéreleménének kulcsát. Ennek segítségével a fa rekurzívan felépíthető.

Hátizsák feladat

Adottak a $t_1, t_2, \ldots, t_n$ tárgyak. A t_i tárgy súlya w_i, értéke v_i, ahol w_i és v_i pozitív egész számok. Kiválasztandó a tárgyaknak egy olyan részhalmaza, amelyben a tárgyak értékének összege a lehető legnagyobb, súlyuk összege viszont nem halad meg egy adott W pozitív egész számot (a hátizsák kapacitását)!

A feladatot dinamikus programozással oldjuk meg. Minden $1 \leqslant i \leqslant n$ és $1 \leqslant j \leqslant W$ esetén jelölje $c[i,j]$ azon feladat egy optimális megoldásában a tárgyak értékeinek az összegét, amikor a hátizsák kapacitása j és a $t_1, t_2, \ldots, t_i$ tárgyak közül választhatunk. A formulák kompakt felírhatóságának érdekében legyen $c[i,j] = 0$, ha $i = 0$ vagy $j = 0$.

Legyen most $1 \leqslant i \leqslant n$ és $1 \leqslant j \leqslant W$, továbbá legyen $\mathcal{O}$ egy optimális megoldása annak a feladatnak, amikor a hátizsák kapacitása j és a $t_1, t_2, \ldots, t_i$ tárgyak közül választhatunk. Ha $w_i > j$, akkor t_i nyilván nem szerepelhet $\mathcal{O}$-ban, így $\mathcal{O}$ szükségképpen egy optimális megoldása annak a feladatnak, amikor a hátizsák kapacitása j és a $t_1, t_2, \ldots, t_{i-1}$ tárgyak közül választhatunk. Ebben az esetben az összérték $c[i-1,j]$. Tegyük fel ezután, hogy $w_i \leqslant j$.

- Ha t_i szerepel az $\mathcal{O}$-beli tárgyak között, akkor $\mathcal{O}' = \mathcal{O} \setminus \{t_i\}$ egy optimális megoldása annak a feladatnak, amikor a hátizsák kapacitása $j - w_i$ és a $t_1, t_2, \ldots, t_{i-1}$ tárgyak közül választhatunk. Valóban, ha lenne $\mathcal{O}'$-nél nagyobb összértékű megoldása annak a feladatnak, amikor a hátizsák kapacitása $j - w_i$ és a $t_1, t_2, \ldots, t_{i-1}$ tárgyak közül választhatunk, akkor ehhez hozzávéve a t_i tárgyat egy $\mathcal{O}$-nál nagyobb összértékű megoldását kapnánk annak a feladatnak, amikor a hátizsák kapacitása j és a $t_1, t_2, \ldots, t_i$ tárgyak közül választhatunk, ellentmondás. Az összérték ekkor $v_i + c[i-1, j-w_i]$.

- Ha t_i nem szerepel az $\mathcal{O}$-beli tárgyak között, akkor $\mathcal{O}$ nyilván egy optimális megoldása annak a feladatnak, mikor a hátizsák kapacitása j és a $t_1, t_2, \ldots, t_{i-1}$ tárgyak közül választhatunk. Az összérték ekkor $c[i-1, j]$.

Mivel nem tudjuk, hogy t_i szerepel-e az $\mathcal{O}$-beli tárgyak között, ezért mindkét lehetőséget megvizsgáljuk, és a kedvezőbbet választjuk. Így

$$c[i,j] = \begin{cases} c[i-1,j] & \text{ha } w_i > j, \\ \max(v_i + c[i-1, j-w_i], c[i-1,j]) & \text{ha } j \geqslant w_i. \end{cases}$$

Annak érdekében, hogy tárgyak egy optimális megoldást adó részhalmazát is meg tudjuk adni, még egy jelölést vezetünk be. Minden $1 \leqslant i \leqslant n$ és $1 \leqslant j \leqslant W$ esetén legyen $d[i,j]$ a `TRUE` logikai érték, ha $c[i,j]$ meghatározásánál azt találtuk, hogy a t_i tárgy beválasztásával nagyobb értékű megoldást kapunk, mint anélkül, egyébként legyen $d[i,j]$ a `FALSE` logikai érték.

```
Hátizsák(n,v,w,W)
for j=0 to W do
  c[0,j]=0
for i=1 to n do
  c[i,0]=0
  d[i,0]=FALSE
  for j=1 to W do
    if w[i]>j
      then
        c[i,j]=c[i-1,j]
        d[i,j]=FALSE
      else
        if v[i]+c[i-1,j-w[i]]>c[i-1,j]
          then
            c[i,j]=v[i]+c[i-1,j-w[i]]
```

```
            d[i,j]=TRUE
          else
            c[i,j]=c[i-1,j]
            d[i,j]=FALSE
return c,d
```

A $c[i,j]$ értékek kiszámítását úgy tekinthetjük, mintha egy $(n+1)\times(W+1)$ méretű táblázatot töltenénk ki felülről lefelé, soronként balról jobbra. A feladat optimális megoldásának értéke $c[n,W]$.

Minden $c[i,j]$ érték konstans költséggel számítható, így az algoritmus teljes költsége $O(nW)$. Jegyezzük meg, hogy az algoritmus nem polinomiális!

A d tömb felhasználásával tárgyak egy optimális megoldást adó részhalmazát a következőképpen határozhatjuk meg:

```
TárgyakNyomtatása
j=W
for i=n downto 1 do
  if d[i,j]=TRUE then
    print i '. tárgy'
    j=j-w[i]
```

Hátizsák feladat még egyszer

A hátizsák feladat megoldására mutatunk egy alternatív, szintén dinamikus programozás algoritmust. A részproblémákat kicsit másképp fogjuk definiálni: minden $1 \leqslant i \leqslant n$ és $1 \leqslant j \leqslant v_1 + v_2 + \cdots + v_i$ esetén keressük a $t_1, t_2, \ldots, t_i$ tárgyak egy olyan részhalmazát, amelyben a tárgyak összértéke legalább j, összsúlya pedig a lehető legkisebb. Jelölje $c[i,j]$ az összsúlyt ennek a részproblémának az optimális megoldásában. Bár ezen részproblémák egyike se egyezik meg az eredeti hátizsák feladattal, annak megoldása egyszerűen megkapható ezekből: a legnagyobb olyan j érték, amelyre $c[n,j] \leqslant W$. A formulák kompakt felírhatóságának érdekében legyen $c[i,0] = 0$ minden $i = 0, 1, \ldots, n$ esetén.

Legyen most $1 \leqslant i \leqslant n$ és $1 \leqslant j \leqslant v_1 + v_2 + \cdots + v_i$, továbbá legyen $\mathcal{O}$ egy optimális megoldása annak a feladatnak, amikor az elérni kívánt érték j és a $t_1, t_2, \ldots, t_i$ tárgyak közül választhatunk. Ha $j > v_1 + v_2 + \cdots + v_{i-1}$, akkor t_i biztosan szerepel az $\mathcal{O}$-beli tárgyak között, és $\mathcal{O}' = \mathcal{O} \setminus \{t_i\}$ egy optimális megoldása annak a feladatnak, amikor az elérni kívánt érték $\max(0, j - v_i)$ és a $t_1, t_2, \ldots, t_{i-1}$ tárgyak közül választhatunk. Valóban, ha lenne $\mathcal{O}'$-nél

kisebb összsúlyú megoldása ennek a feladatnak, akkor ehhez hozzávéve a t_i tárgyat egy $\mathcal{O}$-nál kisebb összsúlyú megoldását kapnánk annak a feladatnak, amikor az elérni kívánt érték j és a $t_1, t_2, \ldots, t_i$ tárgyak közül választhatunk, ellentmondás. Így ekkor $c[i,j] = w_i + c[i-1, \max(0, j-v_i)]$. Tegyük fel ezután, hogy $j \leqslant v_1 + v_2 + \cdots + v_{i-1}$. Most két eset van.

- Ha t_i szerepel az $\mathcal{O}$-beli tárgyak között, akkor $\mathcal{O}' = \mathcal{O} \setminus \{t_i\}$ egy optimális megoldása annak a feladatnak, amikor az elérni kívánt érték $\max(0, j-v_i)$ és a $t_1, t_2, \ldots, t_{i-1}$ tárgyak közül választhatunk. Ismét, ha lenne $\mathcal{O}'$-nél kisebb összsúlyú megoldása ennek a feladatnak, akkor ehhez hozzávéve a t_i tárgyat egy $\mathcal{O}$-nál kisebb összsúlyú megoldását kapnánk annak a feladatnak, amikor az elérni kívánt érték j és a $t_1, t_2, \ldots, t_i$ tárgyak közül választhatunk, ellentmondás. Az összsúly ekkor $w_i + c[i-1, \max(0, j-v_i)]$.

- Ha t_i nem szerepel az $\mathcal{O}$-beli tárgyak között, akkor $\mathcal{O}$ nyilván egy optimális megoldása annak a feladatnak, mikor az elérni kívánt érték j és a $t_1, t_2, \ldots, t_{i-1}$ tárgyak közül választhatunk. Az összsúly ekkor $c[i-1, j]$.

Mivel nem tudjuk, hogy t_i szerepel-e az $\mathcal{O}$-beli tárgyak között, ezért mindkét lehetőséget megvizsgáljuk, és a kedvezőbbet választjuk. Így ekkor

$$c[i,j] = \min(c[i-1, j], w_i + c[i-1, \max(0, j-v_i)]).$$

A $c[i,j]$ értékek kiszámítását úgy tekinthetjük, mintha egy $(n+1) \times (v_1 + v_2 + \cdots + v_n + 1)$ méretű táblázatot töltenénk ki felülről lefelé, soronként balról jobbra. Minden $c[i,j]$ érték konstans költséggel számítható, így ha bevezetjük a $v^* = \max_i v_i$ jelölést, akkor $v_1 + v_2 + \cdots + v_n \leqslant nv^*$ miatt az algoritmus teljes költsége $O(n^2 v^*)$.

Ha a $c[i,j]$ értékek számításánál azt is feljegyezzük, hogy kedvezőbb volte a t_i tárgyat kiválasztani vagy nem, akkor magát egy optimális megoldást is könnyen rekonstruálhatunk.

Ezek után megmutatjuk, hogy a hátizsák feladatra tetszőleges ε pozitív valós számhoz van olyan polinomiális algoritmus, amely által szolgáltatott megoldás értéke az optimális megoldás értékének legalább $1/(1+\varepsilon)$-szorosa.

Feltehetjük, hogy minden tárgy súlya legfeljebb W, amelyekre ez nem teljesül, azok biztos nem lehetnek benne az optimális megoldásban (valójában semmilyen megoldásban), így ezeket elhagyhatjuk. Legyen ε pozitív valós szám. Az egyszerűség kedvéért tegyük fel, hogy ε reciproka egész szám.

Legyen $b = \frac{\varepsilon}{2n} \max_i v_i$, és tekintsük azt a feladatot, amikor minden $1 \leqslant i \leqslant n$ esetén a t_i tárgy értéke $\tilde{v}_i = \lceil v_i/b \rceil b$, súlya változatlanul w_i, és a

hátizsák kapacitása is ugyanúgy W. Jegyezzük meg, hogy $v_i \leqslant \tilde{v}_i \leqslant v_i + b$ minden $1 \leqslant i \leqslant n$ esetén, azaz a v_i és $\tilde{v}_i$ értékek elég közel vannak egymáshoz, ami a különbség, hogy a $\tilde{v}_i$ értékek b egész számú többszörösei. Legyen még $\hat{v}_i = \tilde{v}_i/b$ minden $1 \leqslant i \leqslant n$ esetén. Vegyük észre, hogy a hátizsák feladatnak egy optimális megoldása a $\tilde{v}_i$ értékekkel optimális megoldás lesz a $\hat{v}_i$ értékekkel is, és viszont, a különbség csupán az összértékben van, az utóbbinak b-szerese az előbbi. Futtassuk most a fenti algoritmust a v_i értékek helyett a $\hat{v}_i$ értékekkel, és adjuk vissza azokat a tárgyakat, amelyeket az algoritmus beválasztott a megoldásba.

Jelölje a megoldásba beválasztott tárgyak indexeinek a halmazát I. Először is jegyezzük meg, hogy $\sum_{i \in I} w_i \leqslant W$, hiszen a súlyokon nem változtattunk.

Az algoritmus költségének meghatározásához szükségünk van a $\max_i \hat{v}_i$ értékre. Vegyük észre, hogy ha $v_j = \max_i v_i$, akkor $\hat{v}_j = \max_i \hat{v}_i$, így

$$\max_i \hat{v}_i = \hat{v}_j = \lceil v_j/b \rceil = 2n\varepsilon^{-1}.$$

Következésképpen az algoritmus költsége $O(n^3\varepsilon^{-1})$, ami polinomiális n-ben minden rögzített $\varepsilon > 0$ esetén.

Vizsgáljuk meg milyen messze van az algoritmus által adott megoldás az optimumtól! Tekintsük a tárgyaknak egy tetszőleges olyan $\{t_i \mid i \in I^*\}$ részhalmazát, amelyre $\sum_{i \in I^*} w_i \leqslant W$. Mivel az algoritmus által adott megoldás a $\tilde{v}_i$ értékekkel optimális, ezért

$$\sum_{i \in I} \tilde{v}_i \geqslant \sum_{i \in I^*} \tilde{v}_i,$$

és így

$$\sum_{i \in I^*} v_i \leqslant \sum_{i \in I^*} \tilde{v}_i \leqslant \sum_{i \in I} \tilde{v}_i \leqslant \sum_{i \in I} (v_i + b) \leqslant nb + \sum_{i \in I} v_i.$$

Ha most ismét $v_j = \max_i v_i$, akkor egyrészt $v_j = 2\varepsilon^{-1}nb$, másrészt $v_j = \tilde{v}_j$. Feltételünk szerint $w_i \leqslant W$ minden $1 \leqslant i \leqslant n$ esetén, ennélfogva

$$\sum_{i \in I} \tilde{v}_i \geqslant \tilde{v}_j = 2\varepsilon^{-1}nb.$$

A fenti egyenlőtlenségláncból kiolvashatjuk, hogy

$$\sum_{i \in I} v_i \geqslant \sum_{i \in I} \tilde{v}_i - nb,$$

ahonnan

$$\sum_{i \in I} v_i \geqslant (2\varepsilon^{-1} - 1)nb$$

adódik. Ezt átrendezve kapjuk, hogy

$$nb \leqslant \varepsilon \sum_{i \in I} v_i$$

tetszőleges $\varepsilon \leqslant 1$ pozitív számra, következésképpen

$$\sum_{i \in I^*} v_i \leqslant \sum_{i \in I} v_i + nb \leqslant (1 + \varepsilon) \sum_{i \in I} v_i,$$

amit bizonyítani akartunk.

Részhalmaz összeg

Adottak a $w_1, w_2, \ldots, w_n, W$ természetes számok. Döntsük el, hogy

$$\sum_{i \in I} w_i = W$$

teljesül-e valamilyen $I \subseteq \{1, 2, \ldots, n\}$ indexhalmaz esetén!

Ez a hátizsák feladat speciális esete. A t_i tárgy súlya és értéke is legyen w_i minden $1 \leqslant i \leqslant n$ esetén, valamint legyen a hátizsák kapacitása W. Ha a feladat optimális megoldásában a tárgyak összértéke W, akkor a részhalmaz összeg feladatnak van megoldása, ha kisebb, akkor nincs.

Mohó algoritmusok

Egy optimalizálási feladat megoldására kifejlesztett algoritmus tipikusan olyan lépések sorozatából áll, ahol minden lépésben több lehetőség közül kell valamilyen szempont szerint választanunk. Sok esetben a dinamikus programozás alkalmazása olyan, mintha ágyúval lőnénk verébre; egyszerűbb és hatékonyabb algoritmus is szerkeszthető. Egy mohó algoritmus úgy próbálja megtalálni egy probléma optimális megoldását, hogy minden döntési pontban azt a lépést választja, amely az adott pillanatban a legjobbnak tűnik. Ez a heurisztika természetesen nem mindig vezet el az optimális megoldáshoz, mindazonáltal számos probléma megoldható mohó algoritmussal. Az algoritmus kifejlesztésekor legtöbbször a következő lépéseket hajtjuk végre:

(1) Fogalmazzuk meg a problémát úgy, hogy minden egyes választás után csak egy megoldandó részprobléma keletkezzen.

(2) Bizonyítsuk be, hogy mindig van olyan optimális megoldása az eredeti problémának, amely tartalmazza a mohó választást (mohó választás tulajdonság).

(3) Mutassuk meg, hogy a mohó választással olyan részprobléma keletkezik, amelynek optimális megoldásához hozzávéve a mohó választást, az eredeti probléma egy optimális megoldását kapjuk (optimális részstruktúra tulajdonság).

Mely optimalizálási problémáknak van mohó algoritmusú megoldása? Erre a kérdésre általános válasz nem létezik, de a mohó választás tulajdonság és az optimális részstruktúra tulajdonság két kulcsfontosságú összetevő. Ha meg tudjuk mutatni, hogy egy feladat rendelkezik e két tulajdonsággal, akkor nagy eséllyel ki tudunk fejleszteni mohó algoritmusú megoldást.

Átlagos várakozási idő

Egy étterembe a déli nyitáskor n vendég érkezik. A vendégek jól ismerik a választékot, így azonnal rendelnek. A V_i vendég által rendelt étel elkészí-

tése t_i ideig tart. A szakács egyszerre csak egy étellel tud foglalkozni, és ha valamelyiket elkezdte, akkor azt addig nem hagyja abba, amíg teljesen el nem készült vele. Ha elkészült egy étel, azt rögtön felszolgálják a megfelelő vendégnek. Adjunk hatékony algoritmust annak eldöntésére, hogy a szakács milyen sorrendben készítse el az ételeket, ha azt szeretnénk, hogy a w_i várakozási idők átlaga a lehető legkisebb legyen!

Például ha három vendég van, akik olyan ételeket rendeltek, amelyek elkészítési ideje $t_1 = 2$, $t_2 = 4$, $t_3 = 3$, és a rendeléseket a $(3, 1, 2)$ sorrendben teljesíti az étterem, akkor a várakozási idők $w_3 = 3$, $w_1 = 5$, $w_2 = 9$, az átlagos várakozási idő pedig $17/3$.

Az algoritmus egyszerű: a szakács az elkészítési idő szerint monoton növekvő sorrendben készítse el az ételeket. Az algoritmus helyessége a következő két állításból adódik.

Állítás. Az ételek elkészítésének van olyan optimális ütemezése, amely szerint a legrövidebb elkészítési idejű ételt készítik el elsőnek.

Bizonyítás. Jelölje $i_{\min}$ valamelyik legrövidebb elkészítési idő indexét, és legyen $\sigma\colon \{1, 2, \ldots, n\} \to \{1, 2, \ldots, n\}$ az ételek elkészítésének egy optimális ütemezése. Ha $\sigma(1) = i_{\min}$, akkor kész vagyunk. Ezért tegyük fel, hogy $\sigma(1) \neq i_{\min}$.

Megmutatjuk, hogy az ételek elkészítésének van olyan σ' optimális ütemezése, amelyben $\sigma'(1) = i_{\min}$. Legyen

$$\sigma'\colon \{1, 2, \ldots, n\} \to \{1, 2, \ldots, n\},\ i \mapsto \begin{cases} i_{\min} & \text{ha } i = 1, \\ \sigma(1) & \text{ha } i = i^*, \\ \sigma(i) & \text{különben,} \end{cases}$$

ahol $i_{\min} = \sigma(i^*)$. A σ ütemezés mellett a várakozási idők

$$w_{\sigma(i)} = \sum_{j=1}^{i} t_{\sigma(j)}$$

minden $1 \leqslant i \leqslant n$ esetén, az átlagos várakozási idő pedig

$$W(\sigma) = \frac{1}{n} \sum_{i=1}^{n} w_{\sigma(i)} = \frac{1}{n} \sum_{i=1}^{n} \sum_{j=1}^{i} t_{\sigma(j)} = \frac{1}{n} \sum_{i=1}^{n} (n - i + 1) t_{\sigma(i)}.$$

Hasonlóan, a σ' ütemezés mellett a várakozási idők

$$w_{\sigma'(i)} = \sum_{j=1}^{i} t_{\sigma'(j)}$$

minden $1 \leqslant i \leqslant n$ esetén, az átlagos várakozási idő pedig

$$W(\sigma') = \frac{1}{n}\sum_{i=1}^{n} w_{\sigma'(i)} = \frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{i} t_{\sigma'(j)} = \frac{1}{n}\sum_{i=1}^{n}(n-i+1)t_{\sigma'(i)}.$$

Vessük össze a σ és σ' ütemezések melletti átlagos várakozási időket:

$$W(\sigma') - W(\sigma) = \frac{1}{n}\sum_{i=1}^{n}(n-i+1)\left(t_{\sigma'(i)} - t_{\sigma(i)}\right) =$$
$$\frac{1}{n}\left(n\left(t_{\sigma'(1)} - t_{\sigma(1)}\right) + (n-i^*+1)\left(t_{\sigma'(i^*)} - t_{\sigma(i^*)}\right)\right) =$$
$$\frac{1}{n}\left(n\left(t_{i_{\min}} - t_{\sigma(1)}\right) + (n-i^*+1)\left(t_{\sigma(1)} - t_{i_{\min}}\right)\right) =$$
$$\frac{1}{n}(i^*-1)\left(t_{i_{\min}} - t_{\sigma(1)}\right) \leqslant 0,$$

hiszen $i^* > 1$ és $t_{i_{\min}} \leqslant t_{\sigma(1)}$. Mivel σ egy optimális ütemezés, így $W(\sigma') < W(\sigma)$ nem lehetséges. Ennélfogva $W(\sigma') = W(\sigma)$, ami azt jelenti, hogy σ' is egy optimális ütemezés.

Állítás. Tegyük fel, hogy az ételek elkészítésének van olyan optimális ütemezése, amely szerint a V_k vendég által rendelt ételt készítik el elsőnek. Tegyük fel azt is, hogy

$$\sigma'\colon \{1,2,\ldots,n-1\} \to \{1,\ldots,k-1,k+1,\ldots,n\}$$

a többi étel elkészítésének egy optimális ütemezése. Ekkor

$$\sigma\colon \{1,2,\ldots,n\} \to \{1,2,\ldots,n\},\ i \mapsto \begin{cases} k & \text{ha } i = 1, \\ \sigma'(i-1) & \text{ha } 2 \leqslant i \leqslant n \end{cases}$$

az összes étel elkészítésének egy optimális ütemezése.

Bizonyítás. Indirekt tegyük fel, hogy σ nem optimális ütemezése az összes étel elkészítésének. Legyen $\pi\colon \{1,2,\ldots,n\} \to \{1,2,\ldots,n\}$ az ételek elkészítésének egy olyan optimális ütemezése, amely szerint a V_k vendég által rendelt ételt készítik el elsőnek, azaz amelyre $\pi(1) = k$. Ekkor a π ütemezéshez tartozó $W(\pi)$ átlagos várakozási idő kisebb, mint a σ ütemezéshez tartozó $W(\sigma)$ átlagos várakozási idő.

Tekintsük most a V_k vendég által rendelt ételen kívüli ételek elkészítésének a

$$\pi'\colon \{1,2,\ldots,n-1\} \to \{1,\ldots,k-1,k+1,\ldots,n\},\ i \mapsto \pi(i+1)$$

ütemezését. Vessük össze a π' és σ' ütemezések melletti átlagos várakozási időket:

$$W(\pi') - W(\sigma') = \frac{1}{n-1}\sum_{i=1}^{n-1} w_{\pi'(i)} - \frac{1}{n-1}\sum_{i=1}^{n-1} w_{\sigma'(i)} =$$

$$\frac{n}{n-1}\left(\frac{1}{n}\left(t_k + \sum_{i=1}^{n-1}\left(t_k + w_{\pi'(i)}\right)\right) - \frac{1}{n}\left(t_k + \sum_{i=1}^{n-1}\left(t_k + w_{\sigma'(i)}\right)\right)\right) =$$

$$\frac{n}{n-1}\left(\frac{1}{n}\sum_{i=1}^{n} w_{\pi(i)} - \frac{1}{n}\sum_{i=1}^{n} w_{\sigma(i)}\right) = \frac{n}{n-1}\left(W(\pi) - W(\sigma)\right) < 0,$$

ami ellentmondás, hiszen σ' egy optimális ütemezése a V_k vendég által rendelt ételen kívüli ételek elkészítésének. Következésképpen σ az összes étel elkészítésének egy optimális ütemezése.

Az algoritmus helyességének belátására mutatunk egy alternatív módszert is, amely szintén gyakran alkalmazható. Az egyszerűség kedvéért tegyük fel, hogy a rendelések már az elkészítési idejük szerint monoton növekvően rendezettek (ha ez nem teljesül, akkor először rendezzük őket):

$$t_1 \leqslant t_2 \leqslant \cdots \leqslant t_n.$$

A fenti algoritmus ekkor a következőképpen ütemezi az ételek elkészítését: a V_1 vendég által rendelt étel elkészítésének kezdési ideje $s_1 = 0$, befejezési ideje $f_1 = s_1 + t_1$, a V_2 vendég által rendelt étel elkészítésének kezdési ideje $s_2 = f_1$, befejezési ideje $f_2 = s_2 + t_2$, és így tovább.

Belátjuk, hogy ez az ütemezés optimális. Jelölje $\mathcal{P}$ az algoritmus által előállított ütemezést. Jegyezzük meg, hogy ebben nincs üresjárat, amikor a szakács nem foglalkozik egyik étel elkészítésével sem, miközben vannak még elkészítésre várók.

Nyilvánvaló, hogy az optimális ütemezések között is kell lenni üresjárat nélkülinek. Jelöljön $\mathcal{O}$ egy ilyen optimális ütemezést. Ha $\mathcal{O} = \mathcal{P}$, akkor készen vagyunk. Tegyük fel ezért, hogy $\mathcal{O} \neq \mathcal{P}$. Ekkor az $\mathcal{O}$ ütemezésben szükségképpen vannak olyan V_i és V_j vendégek, akik által rendelt ételeket közvetlenül egymás után készíti el a szakács, és amelyekre $i > j$.

Tekintsük most azt az $\mathcal{O}'$ ütemezést, amelyet $\mathcal{O}$-ból a V_i és V_j vendégek által rendelt ételek elkészítésének felcserélésével kapunk. Vizsgáljuk meg, hogy a cserével növekedhetett-e az átlagos várakozási idő, vagy ami ezzel ekvivalens, a várakozási idők összege!

Csak a V_i és V_j vendégek által rendelt ételek elkészítésének ütemezése változott meg, így a többi vendég várakozási ideje a csere után ugyanannyi, mint a csere előtt volt. A csere után a V_i vendég által rendelt étel nyilván ugyankkor készül el, mint V_j vendégé a csere előtt. Másrészt a V_j vendég által rendelt étel a csere után mindenképp elkészül addigra, mint a V_i vendégé a csere előtt, hiszen a V_j vendég által rendelt étel elkészítési ideje nem haladja meg a V_i vendég által rendelt étel elkészítési idejét (ne feledjük $j < i$). Így az elkészítési idők összege, és ennélfogva az átlaga az $\mathcal{O}'$ ütemezésnél nem lehet nagyobb, mint az $\mathcal{O}$ ütemezésnél.

Algoritmusunk helyessége innen már egyszerűen adódik. Az optimális $\mathcal{O}$ ütemezésből legfeljebb $\binom{n}{2}$ szomszédos étel elkészítésének cseréjével eljuthatunk a $\mathcal{P}$ ütemezéshez (vö. buborék rendezés). Mivel az átlagos várakozási idő egyik lépésben sem növekszik, így $\mathcal{P}$ szükségképpen szintén egy optimális ütemezés.

Minimális késés

Tegyük fel, hogy adott feladatok egy $\mathcal{A} = \{a_1, a_2, \ldots, a_n\}$ halmaza, amelyek végrehajtása egy közös erőforrás használatával lehetséges csak. Az erőforrás egyszerre egy feladattal tud foglalkozni, és ha valamelyiket elkezdte, akkor azt addig nem hagyja abba, amíg teljesen el nem készült vele. Minden a_i feladathoz adott a t_i végrehajtási ideje, és egy d_i határidő, amelyre a feladat végrehajtásával el kellene készülni.

Tegyük fel, hogy minden feladatot végre kell hajtanunk, de az megengedett, hogy bizonyosakat a határidejük után fejezzünk csak be. Minden a_i feladathoz rendeljük hozzá azt a t_i hosszú intervallumot, amikor az erőforrást a feladat végrehajtására akarjuk használni. Jelölje ezt az intervallumot $[s_i, f_i]$, ahol $s_i \geqslant 0$ és $f_i = s_i + t_i$ természetes módon. Mivel az erőforrás egyszerre egy feladattal tud csak foglalkozni, ezeknek az intervallumoknak a végpontjaiktól eltekintve páronként diszjunktaknak kell lenni.

Azt fogjuk mondani, hogy egy a_i feladatot késve hajtunk végre, ha $f_i > d_i$. Ebben az esetben jelölje $l_i = d_i - f_i$ a késés nagyságát. Ha egy a_i feladat a határidején belül végrehajtásra kerül, akkor legyen $l_i = 0$ definíció szerint.

Adjunk hatékony algoritmust a feladatok egy olyan ütemezésének megkeresésére, amelynél az $L = \max\{l_i \mid 1 \leqslant i \leqslant n\}$ maximális késés a lehető legkevesebb!

Tegyük fel, hogy a feladatok a határidejük szerint monoton növekvően

rendezettek (ha ez nem teljesül, akkor először rendezzük őket):

$$d_1 \leqslant d_2 \leqslant \cdots \leqslant d_n.$$

Ütemezzük a feladatokat ebben a sorrendben: legyen a_1 kezdési ideje $s_1 = 0$, befejezési ideje $f_1 = s_1 + t_1$, legyen a_2 kezdési ideje $s_2 = f_1$, befejezési ideje $f_2 = s_2 + t_2$, és így tovább. A költség nyilván $O(n \log n)$.

Belátjuk, hogy ez az ütemezés optimális. Jelölje $\mathcal{P}$ az algoritmus által előállított ütemezést. Jegyezzük meg, hogy ebben nincs üresjárat, amikor az erőforrás nem foglalkozik egyik feladattal sem, miközben vannak még elvégzésre várók.

Nyilvánvaló, hogy az optimális ütemezések között is kell lenni üresjárat nélkülinek. Jelöljön $\mathcal{O}$ egy ilyen optimális ütemezést. Ha $\mathcal{O} = \mathcal{P}$, akkor készen vagyunk. Tegyük fel ezért, hogy $\mathcal{O} \neq \mathcal{P}$. Ekkor az $\mathcal{O}$ ütemezésben szükségképpen vannak olyan a_i és a_j közvetlenül egymás után következő feladatok, amelyekre $i > j$.

Tekintsük most azt az $\mathcal{O}'$ ütemezést, amelyet $\mathcal{O}$-ból az a_i és a_j feladatok felcserélésével kapunk. Vizsgáljuk meg, hogy a cserével növekedhetett-e a maximális késés!

Csak az a_i és a_j feladatok ütemezése változott meg, így a többi feladatnál a késés a csere után ugyanannyi, mint a csere előtt volt. A cserével az a_j feladat előbbre került, így a késése nyilván nem növekedett.

Nézzük ezután az a_i feladatot. Jelölje a_j befejezési idejét az $\mathcal{O}$ ütemezésnél f_j, késését pedig l_j. Ekkor a_i befejezési ideje az $\mathcal{O}'$ ütemezésnél $f'_i = f_j$. Jelölje a_i késését az $\mathcal{O}'$ ütemezésnél l'_i.

Ha $l'_i = 0$, vagyis a_i nem késik az $\mathcal{O}'$ ütemezésnél, akkor nyilván nem késhetett az $\mathcal{O}$ ütemezésnél sem, hiszen ott korábbra volt ütemezve. Ha pedig $l'_i > 0$, akkor $d_i \geqslant d_j$ figyelembe vételével

$$l'_i = f'_i - d_i = f_j - d_i \leqslant f_j - d_j = l_j$$

adódik, vagyis a_i késése az $\mathcal{O}'$ ütemezésnél nem lehet nagyobb, mint a_j késése az $\mathcal{O}$ ütemezésnél. Mindebből következik, hogy a maximális késés az $\mathcal{O}'$ ütemezésnél nem lehet nagyobb, mint az $\mathcal{O}$ ütemezésnél.

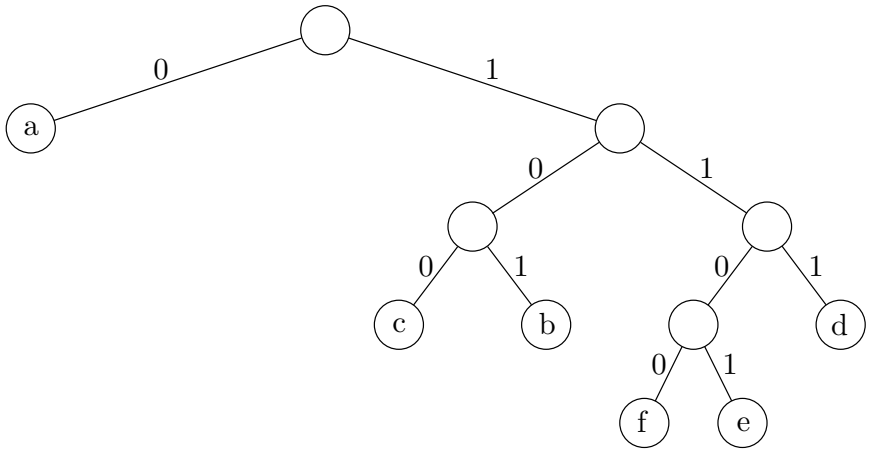
Algoritmusunk helyessége innen már egyszerűen adódik. Az optimális $\mathcal{O}$ ütemezésből legfeljebb $\binom{n}{2}$ szomszédos feladat cseréjével eljuthatunk a $\mathcal{P}$ ütemezéshez (vö. buborék rendezés). Mivel a maximális késés egyik lépésben sem növekszik, így $\mathcal{P}$ szükségképpen szintén egy optimális ütemezés.

Információtömörítés

Tegyük fel, hogy egy n különböző karakterből álló adatállományt akarunk tömörítetten, bitsorozatként tárolni. Ha fix hosszú kódszavakat használunk, akkor $\lceil \log_2 n \rceil$ bitre van szükség az n-féle karakter tárolására. Így egy m hosszú állomány tárolása összesen $m\lceil \log_2 n \rceil$ bitet igényel. Van ennél hatékonyabb módszer?

Ha megengedünk eltérő hosszú kódszavakat, akkor probléma lehet a dekódolással, vagyis az eredeti állománynak a kódolt bitsorozatból való visszanyerésével. Egy lehetséges megoldást kínál a következő. Egy bitsorozatokból álló kód prefix kód, ha egyik karakter kódja sem prefixe más karakter kódjának. Formálisan, ha x és y két különböző karakter kódja, akkor nincs olyan z bitsorozat, amelyre $xz = y$. Itt xz azt a bitsorozatot jelöli, melyben x bitjeit z bitjei követik. Egy prefix kóddal leírt állomány egyértelműen dekódolható. Az állomány elejétől addig megyünk a bitek mentén, amíg egy karakter kódszavát kapjuk. A prefix feltétel miatt csak ez lehet az első karakter. Ezt az eljárást addig folytatjuk, amíg a kódolt állomány végére nem érünk.

Egy prefix kódhoz természetes módon hozzárendelhetünk egy bináris fát. A fa levelei a kódolandó karakterek. Egy adott karakter kódját a gyökértől a karakterig vezető út ábrázolja: a 0 azt jelenti, hogy balra megyünk, az 1 pedig azt, hogy jobbra megyünk az úton a fában.



Ha adott egy prefix kód T fája, akkor egyszerűen kiszámítható az adatállomány kódolásához szükséges bitek száma. A C ábécé minden c karakterére jelölje $f[c]$ a c karakter előfordulási gyakoriságát az állományban, $d_T(c)$ pedig a c-t tartalmazó levél mélységét a T fában (ami éppen a c karakter kódjának

hossza). A kódoláshoz szükséges bitek száma ekkor

$$B_T(C) = \sum_{c \in C} f[c] d_T(c).$$

Ezt az értéket a fa költségének fogjuk nevezni.

A feladat ezek után egy minimális költségű prefix kód előállítása a karakterek előfordulási gyakoriságának ismeretében.

Először is vegyük észre, hogy egy adatállomány optimális prefix kódját mindig teljes bináris fa ábrázolja, azaz olyan fa, amelyben minden nem levél csúcsnak két gyermeke van. Valóban, ha lenne olyan belső csúcs, amelynek csak egy gyereke van, akkor ezt a csúcsot törölve (egyetlen gyerekét a helyére téve) a fa költségét csökkenteni tudnánk. Innen egyszerűen következik, hogy az optimális prefix kód fájának $|C|$ levele és $|C| - 1$ belső csúcsa van.

A következő algoritmus egy optimális prefix kód fáját konstruálja meg. Az algoritmus alulról felfelé haladva építi meg ezt a fát. Kezdetben $|C|$ számú csúcs van, amelyek mindegyike levél, majd $|C| - 1$ számú "összevonás" végrehajtása után alakul ki a végső fa. Két csúcs összevonásának eredménye egy új csúcs, melynek gyakorisága a két összevont csúcs gyakoriságának az összege. Mindig a két legkisebb gyakoriságú csúcsot vonjuk össze; ezek azonosítására egy f szerint kulcsolt Q kupacot használunk. Az algoritmus a megkonstruált fa gyökerével tér vissza.

```
OptimálisPrefixKód(C,f,n)
Q=KupacotÉpít(C,f,n)
for i=1 to n-1 do
  x=MinTöröl(Q)
  y=MinTöröl(Q)
  z=CsúcsotLétrehoz
  bal[z]=x
  jobb[z]=y
  f[z]=f[x]+f[y]
  Beszúr(Q,z)
return MinTöröl(Q)
```

A kupacépítés költsége $O(n)$. A for ciklus iterációinak száma $n - 1$, így mivel az egyes kupacműveletek költsége $O(\log n)$, a ciklus összköltsége $O(n \log n)$. Ennélfogva az algoritmus teljes költsége $O(n \log n)$. Az algoritmus helyessége a következő két állításból adódik.

Állítás. Legyen x és y a két legkisebb gyakoriságú karakter C-ben. Ekkor van olyan optimális prefix kód, amelyben x és y kódszava ugyanolyan hosszú, és a két kódszó csak az utolsó bitben különbözik.

Bizonyítás. Tekintsünk egy optimális prefix kódot ábrázoló T fát. Legyen a és b ebben a fában a legmélyebben fekvő "testvér" csúcspár. Az általánosság megszorítása nélkül feltehetjük, hogy $f[a] \leqslant f[b]$ és $f[x] \leqslant f[y]$. Mivel $f[x]$ és $f[y]$ a két legkisebb gyakoriság, ezért $f[x] \leqslant f[a]$ és $f[y] \leqslant f[b]$.

Cseréljük fel T-ben az x és a csúcsokat. Jelölje az új fát T'. Ezután cseréljük fel T'-ben az y és b csúcsokat. Jelölje az új fát T''. A T és T' fák költségének különbsége:

$$\begin{aligned} B_T(C) - B_{T'}(C) &= \sum_{c \in C} f[c] d_T(c) - \sum_{c \in C} f[c] d_{T'}(c) \\ &= f[x] d_T(x) + f[a] d_T(a) - f[x] d_{T'}(x) - f[a] d_{T'}(a) \\ &= f[x] d_T(x) + f[a] d_T(a) - f[x] d_T(a) - f[a] d_T(x) \\ &= (f[a] - f[x])(d_T(a) - d_T(x)) \geqslant 0, \end{aligned}$$

hiszen egyrészt az a karakter gyakorisága legalább akkora, mint az x karakteré, másrészt a legalább olyan mélyen van T-ben, mint x. Hasonlóan, a T' és T'' fák költségének különbsége:

$$\begin{aligned} B_{T'}(C) - B_{T''}(C) &= \sum_{c \in C} f[c] d_{T'}(c) - \sum_{c \in C} f[c] d_{T''}(c) \\ &= f[y] d_{T'}(y) + f[b] d_{T'}(b) - f[y] d_{T''}(y) - f[b] d_{T''}(b) \\ &= f[y] d_{T'}(y) + f[b] d_{T'}(b) - f[y] d_{T'}(b) - f[b] d_{T'}(y) \\ &= (f[b] - f[y])(d_{T'}(b) - d_{T'}(y)) \geqslant 0, \end{aligned}$$

hiszen egyrészt a b karakter gyakorisága legalább akkora, mint az y karakteré, másrészt b legalább olyan mélyen van T'-ben, mint y.

Kaptuk tehát, hogy

$$B_T(C) \geqslant B_{T'}(C) \quad \text{és} \quad B_{T'}(C) \geqslant B_{T''}(C),$$

így

$$B_T(C) \geqslant B_{T''}(C).$$

Figyelembe véve, hogy T optimális prefix kódot ábrázoló fa, ez csak úgy lehetséges, ha

$$B_T(C) = B_{T''}(C).$$

Ebből következik, hogy T'' is egy optimális prefix kódot ábrázoló fa, amelyben x és y maximális mélységű testvércsúcsok. Innen az állítás adódik.

Állítás. Legyen x és y a két legkisebb gyakoriságú karakter C-ben. Most tekintsük azt a C' ábécét, amelyet úgy kapunk C-ből, hogy eltávolítjuk x-et és y-t, majd hozzáadunk egy új z karaktert, melyre $f[z] = f[x] + f[y]$. A többi karakter gyakorisága nem változik. Legyen T' egy optimális prefix kódot ábrázoló fa a C' ábécéhez. Ekkor az a T fa, amelyet úgy kapunk T'-ből, hogy a z csúcshoz gyerekként hozzákapcsoljuk x-et és y-t, egy optimális prefix kódot ábrázoló fa a C ábécéhez.

Bizonyítás. Először is jegyezzük meg, hogy

$$\begin{aligned} B_T(C) - B_{T'}(C') &= f[x]d_T(x) + f[y]d_T(y) - f[z]d_{T'}(z) \\ &= (f[x] + f[y])(d_{T'}(z) + 1) - (f[x] + f[y])d_{T'}(z) \\ &= f[x] + f[y]. \end{aligned}$$

Ezek után indirekt tegyük fel, hogy T nem egy optimális prefix kódot ábrázoló fa a C ábécéhez. Legyen T'' egy optimális prefix kódot ábrázoló fa a C ábécéhez. Az előző állítás alapján feltehetjük, hogy x és y testvérek T''-ben. Most

$$B_T(C) > B_{T''}(C).$$

Legyen T''' az a fa, amelyet úgy kapunk T''-ből, hogy eltávolítjuk az x és y csúcsokat, ezek közös szülőjét z-vel jelöljük, és ehhez a z csúcshoz hozzárendeljük az $f[z] = f[x] + f[y]$ gyakoriságot. Ekkor

$$\begin{aligned} B_{T''}(C) - B_{T'''}(C') &= f[x]d_{T''}(x) + f[y]d_{T''}(y) - f[z]d_{T'''}(z) \\ &= (f[x] + f[y])(d_{T'''}(z) + 1) - (f[x] + f[y])d_{T'''}(z) \\ &= f[x] + f[y]. \end{aligned}$$

Így a T''' fára

$$B_{T'''}(C') = B_{T''}(C) - (f[x] + f[y]) < B_T(C) - (f[x] + f[y]) = B_{T'}(C'),$$

ami ellentmond annak, hogy T' egy optimális prefix kódot ábrázoló fa a C' ábécéhez. Ebből következik, hogy T egy optimális prefix kódot ábrázoló fa a C ábécéhez.

Matroidok

A mohó algoritmusokhoz egy szép elmélet kapcsolódik, ami számos esetben alkalmazható annak kimutatására, hogy egy mohó algoritmus optimális megoldást szolgáltat. Az elmélet egy kombinatorikus struktúra fogalmára épül.

Tekintsünk egy S nem üres véges halmazt és S részhalmazainak egy nem üres $\mathcal{F}$ családját. Az $\mathcal{M} = (S, \mathcal{F})$ párt matroidnak nevezzük, ha teljesülnek rá a következők.

(1) Ha $A \in \mathcal{F}$, akkor tetszőleges $B \subseteq A$ esetén $B \in \mathcal{F}$ is fennáll. Speciálisan $\emptyset \in \mathcal{F}$.

(2) Ha $A, B \in \mathcal{F}$ és A elemszáma kisebb, mint B elemszáma, akkor létezik olyan $x \in B \setminus A$ elem, hogy $A \cup \{x\} \in \mathcal{F}$ szintén fennáll.

Az $\mathcal{F}$-beli halmazokat a matroid független halmazainak nevezzük.

Lássunk két példát.

Tekintsünk egy valós számokból álló T mátrixot. Jelölje S a T mátrix oszlopvektorainak halmazát, $\mathcal{F}$ pedig az S olyan részhalmazainak családját, amelyekben a vektorok lineárisan függetlenek. Ekkor $\mathcal{M} = (S, \mathcal{F})$ matroid.

Tekintsünk egy $G = (V, E)$ irányítatlan gráfot. Legyen $S = E$, valamint $\mathcal{F}$ az S olyan részhalmazainak családját, amelyekben az élek erdőt alkotnak. Ekkor $\mathcal{M} = (S, \mathcal{F})$ matroid.

Egy $\mathcal{M} = (S, \mathcal{F})$ matroid $A \in \mathcal{F}$ független halmazát maximálisnak nevezzük, ha tetszőleges $x \in S \setminus A$ elemre $A \cup \{x\} \notin \mathcal{F}$. A matroidokra vonatkozó (2) tulajdonságból azonnal következik, hogy egy matroid minden maximális független halmaza ugyanolyan elemszámú.

Ha egy $\mathcal{M} = (S, \mathcal{F})$ matroid S alaphalmazán adott egy $w\colon S \to \mathbb{R}^+$ súlyfüggvény, akkor súlyozott matroidról beszélünk. A w súlyfüggvényt összegzéssel kiterjeszthetjük az $\mathcal{F}$-beli független részhalmazokra: ha $A \in \mathcal{F}$, akkor legyen

$$w(A) = \sum_{x \in A} w(x).$$

Sok probléma, amelyre a mohó stratégia optimális megoldást szolgáltat, megfogalmazható súlyozott matroid maximális súlyú független halmazának megkereséseként. Megjegyezzük, hogy egy ilyen független halmaz szükségképpen maximális, hiszen a w súlyfüggvény pozitív.

Például a minimális költségű feszítőfa probléma esetén adott egy $G = (V, E)$ súlyozott élű, irányítatlan, összefüggő gráf a w súlyfüggvénnyel, és keressük egy olyan feszítőfáját G-nek, amelyben az élsúlyok összege minimális. Tekintsük a gráfhoz a már fentebb definiált $\mathcal{M} = (E, \mathcal{F})$ matroidot azzal a w'

súlyfüggvénnyel, amelyre $w'(e) = w_0 - w(e)$, ahol w_0 nagyobb, mint a G-beli élsúlyok maximuma. Ebben a súlyozott matroidban minden súly pozitív, és egy optimális halmaz olyan feszítőfa lesz, amely éleinek az eredeti gráfban vett összsúlya minimális. Így minden olyan algoritmus, amely optimális független halmazt tud keresni egy súlyozott matroidban, meg tudja oldani a minimális költségű feszítőfa problémát is.

A következő mohó algoritmus egy ebben az értelemben optimális független halmazt állít elő egy súlyozott matroidban.

```
Mohó(M,w)
A = ∅
H = S
while H ≠ ∅ do
   töröljük H egy legnagyobb súlyú x elemét
   if A ∪ {x} ∈ F then A = A ∪ {x}
return A
```

Dinamikus programozás vs. mohó módszer

Csillagászati megfigyelések

Tegyük fel, hogy adott csillagászati megfigyelések egy $M = \{m_1, m_2, \ldots, m_n\}$ halmaza, amelyeket különböző kutatócsoportok a Hubble űrtávcsővel szeretnének elvégezni. A megfigyelések nem szakíthatók meg és az űrtávcső egy időben legfeljebb egy megfigyelést tud végrehajtani. Minden m_i megfigyelésnek adott az s_i kezdési és az f_i befejezési időpontja, ahol $0 \leqslant s_i < f_i < \infty$. Ezen kívül minden m_i megfigyelésnek adott a w_i tudományos súlya, ahol $w_i > 0$. Az m_i és m_j megfigyeléseket egymást kizáróknak nevezzük, ha az $[s_i, f_i]$ és $[s_j, f_j]$ intervallumoknak van közös belső pontja. Ellenkező esetben azt mondjuk, hogy az m_i és m_j megfigyelések kompatibilisek. Feladatunk páronként kompatibilis M-beli megfigyelések egy olyan $\{m_{i_1}, m_{i_2}, \ldots, m_{i_k}\}$ részhalmazának a meghatározása, amelyre $w_{i_1} + w_{i_2} + \cdots + w_{i_k}$ maximális.

A feladatot dinamikus programozással oldjuk meg. Tegyük fel, hogy a megfigyelések a befejezési idejük szerint monoton növekvően rendezettek (ha ez nem teljesül, akkor először rendezzük őket):

$$f_1 \leqslant f_2 \leqslant \cdots \leqslant f_n.$$

Minden $2 \leqslant i \leqslant n$ esetén legyen $\gamma(i)$ az a legnagyobb $j < i$ index, amelyre az m_j és m_i megfigyelések kompatibilisek. Ha az m_j és m_i megfigyelések egymást kizáróak minden $1 \leqslant j < i$ esetén, akkor legyen $\gamma(i) = 0$, továbbá legyen $\gamma(1) = 0$. A $\gamma(i)$ értékek a megfigyelések befejezési idejeinek monoton növekvően rendezett sorozatában történő bináris kereséssel egyszerűen meghatározhatók.

Minden $1 \leqslant i \leqslant n$ esetén jelölje $v[i]$ azon feladat egy optimális megoldásában a megfigyelések összsúlyát, amikor az $m_1, m_2, \ldots, m_i$ megfigyelések közül választhatunk. A formulák kompakt felírhatóságának érdekében legyen $v[0] = 0$.

Legyen most $1 \leqslant i \leqslant n$ és legyen $\mathcal{O}$ egy optimális megoldása annak a feladatnak, mikor az $m_1, m_2, \ldots, m_i$ megfigyelések közül választhatunk.

- Ha m_i szerepel az $\mathcal{O}$-beli megfigyelések között, akkor $\mathcal{O}' = \mathcal{O} \setminus \{m_i\}$ egy optimális megoldása annak a feladatnak, amikor az $m_1, m_2, \ldots, m_{\gamma(i)}$ események közül választhatunk (ne feledjük, hogy az $m_{\gamma(i)+1}, \ldots, m_{i-2}$, m_{i-1} megfigyelések egymást kizáróak az m_i megfigyeléssel). Valóban, ha lenne $\mathcal{O}'$-nél nagyobb összsúlyú megoldása annak a feladatnak, amikor az $m_1, m_2, \ldots, m_{\gamma(i)}$ megfigyelések közül választhatunk, akkor ehhez hozzávéve az m_i megfigyelést egy $\mathcal{O}$-nál nagyobb összsúlyú megoldását kapnánk annak a feladatnak, mikor az $m_1, m_2, \ldots, m_i$ megfigyelések közül választhatunk, ellentmondás. Az $\mathcal{O}$-beli megfigyelések összsúlya ebben az esetben $w_i + v[\gamma(i)]$.

- Ha m_i nem szerepel az $\mathcal{O}$-beli megfigyelések között, akkor $\mathcal{O}$ nyilván egy optimális megoldása annak a feladatnak, mikor az $m_1, m_2, \ldots, m_{i-1}$ megfigyelések közül választhatunk. Ebben az esetben az $\mathcal{O}$-beli megfigyelések összsúlya $v[i-1]$.

Mivel nem tudjuk, hogy m_i szerepel-e az $\mathcal{O}$-beli megfigyelések között, ezért mindkét lehetőséget megvizsgáljuk és a kedvezőbbet választjuk. Így

$$v[i] = \max(w_i + v[\gamma(i)], v[i-1]).$$

A $v[i]$ értékek kiszámítását úgy tekinthetjük, mintha egy $n+1$ elemű tömböt töltenénk ki balról jobbra. A feladat optimális megoldásában a megfigyelések összsúlya $v[n]$.

Minden $v[i]$ érték $O(1)$ lépésben számítható, így a $v[0:n]$ tömb kitöltésének költsége $O(n)$. Ehhez jön még a kezdeti rendezés és a $\gamma(i)$ értékek meghatározásának költsége, ami $O(n \log n)$. Így az algoritmus teljes költsége $O(n \log n) + O(n) = O(n \log n)$.

Ha minden $1 \leqslant i \leqslant n$ esetén feljegyezzük, hogy $v[i]$ számításánál kedvezőbb volt-e az m_i megfigyelést kiválasztani vagy nem, akkor magát egy optimális megoldást is könnyen rekonstruálhatunk.

Ha minden megfigyelés tudományos súlya ugyanakkora, akkor a feladat egyszerűen páronként kompatibilis M-beli megfigyelések egy maximális elemszámú $\{m_{i_1}, m_{i_2}, \ldots, m_{i_k}\}$ részhalmazának a meghatározása. Erre a feladatra a következő mohó algoritmus is optimális megoldást ad. Tegyük fel ismét, hogy a megfigyelések a befejezési idejük szerint monoton növekvően rendezettek, vagyis $f_1 \leqslant f_2 \leqslant \cdots \leqslant f_n$.

A kiválasztott megfigyeléseket egy H halmazban fogjuk tárolni. Kezdetben legyen $H = \emptyset$. Használni fogunk még egy G listát, amely mindig a még nem vizsgált megfigyeléseket tartalmazza a befejezési idejük szerint monoton növekvően rendezetten. Kezdetben G az összes M-beli megfigyelésből áll. Először rakjuk át G-ből H-ba a legkisebb befejezési időpontú megfigyelést, majd töröljük G-ből az ezzel egymást kizáró megfigyeléseket, amíg egy kompatibilis megfigyelést nem találunk. Ismételjük ezek után a következőt, amíg G-ből nem töröltük az összes megfigyelést. Rakjuk át G-ből H-ba a legkisebb befejezési időpontú megfigyelést, majd töröljük G-ből az ezzel egymást kizáró megfigyeléseket, amíg egy kompatibilis megfigyelést nem találunk.

Legyen $\mathcal{H} = \{m_{j_1}, m_{j_2}, \ldots, m_{j_l}\}$ az algoritmus által kiválasztott megfigyelések halmaza, ahol $j_1 < j_2 < \cdots < j_l$. Világos módon a $\mathcal{H}$-beli megfigyelések páronként kompatibilisek. Belátjuk, hogy $\mathcal{H}$ egy optimális megoldása a feladatnak. Tekintsük a feladat egy $\mathcal{O} = \{m_{i_1}, m_{i_2}, \ldots, m_{i_k}\}$ optimális megoldását, ahol $i_1 < i_2 < \cdots < i_k$. Nyilván $l \leqslant k$.

Először teljes indukcióval megmutatjuk, hogy $f_{j_r} \leqslant f_{i_r}$ minden $1 \leqslant r \leqslant l$ esetén. Ha $r = 1$, akkor ez nyilvánvaló, hiszen az algoritmus elsőként a legkisebb befejezési idejű m_1 megfigyelést választja ki. Legyen most $r > 1$ és tegyük fel, hogy $f_{j_{r-1}} \leqslant f_{i_{r-1}}$. Mivel az $\mathcal{O}$-beli megfigyelések páronként kompatibilisek, ezért $f_{i_{r-1}} \leqslant s_{i_r}$, így $f_{j_{r-1}} \leqslant s_{i_r}$. Ez azt jelenti, hogy az m_{i_r} megfigyelés benne volt abban a halmazban, amelyből az algoritmus a legkisebb befejezési időpontú m_{j_r} megfigyelést választotta, következésképpen $f_{j_r} \leqslant f_{i_r}$.

Indirekt tegyük fel ezután, hogy $\mathcal{H}$ nem egy optimális megoldása a feladatnak, vagyis $l < k$. Az előzőek szerint $f_{j_l} \leqslant f_{i_l}$. Mivel $l < k$, ezért $\mathcal{O}$ tartalmaz egy $f_{i_{l+1}}$ megfigyelést is. Mivel az $\mathcal{O}$-beli megfigyelések páronként kompatibilisek, ezért $f_{i_l} \leqslant s_{i_{l+1}}$, következésképpen $f_{j_l} \leqslant s_{i_{l+1}}$. Ez viszont azt jelenti, hogy miután töröltük G-ből az m_{j_l} megfigyeléssel egymást kizáró megfigyeléseket, az $m_{i_{l+1}}$ megfigyelés még G-ben maradt, így az algoritmus nem fejeződhetett be, ellentmondás. Ebből következik, hogy $\mathcal{H}$ egy optimális megoldása a feladatnak.

Az algoritmus költsége a kezdeti rendezés nélkül $O(n)$, azzal együtt

$$O(n \log n) + O(n) = O(n \log n).$$

Fesztiválbérlet

Egy zenei fesztivál szervezői a fesztiválbérlet árát a következők alapján szeretnék meghatározni. Minden programhoz megállapítanak egy elvi részvételi

díjat, amit teljes egészében ki kellene fizetni egy látogatónak, ha a programon akárcsak részben részt vesz. Legalább mennyit kellene így fizetni egy látogatónak, ha a fesztivál ideje alatt végig részt venne valamilyen programon? Feltehetjük, hogy a fesztiválon soha nincs üresjárat, mindig zajlik legalább egy program.

Formálisan, adott a fesztiválnak az s kezdési és az f befejezési időpontja. Adott továbbá a programok $P = \{p_1, p_2, \ldots, p_n\}$ halmaza. Minden p_i programnak adott az s_i kezdési és az f_i befejezési időpontja, ahol $s \leqslant s_i < f_i \leqslant f$. Feltételünk szerint

$$[s_1, f_1] \cup [s_2, f_2] \cup \cdots \cup [s_n, f_n] = [s, f].$$

Ezen kívül minden p_i programnak adott a w_i részvételi díja, ahol $w_i > 0$. Feladatunk P-beli programok egy olyan $\{p_{i_1}, p_{i_2}, \ldots, p_{i_k}\}$ részhalmazának a meghatározására, amelyre

$$[s_{i_1}, f_{i_1}] \cup [s_{i_2}, f_{i_2}] \cup \cdots \cup [s_{i_k}, f_{i_k}] = [s, f]$$

és $w_{i_1} + w_{i_2} + \cdots + w_{i_k}$ minimális.

A feladatot dinamikus programozással oldjuk meg. Tegyük fel, hogy a programok a kezdési idejük szerint monoton növekvően rendezettek (ha ez nem teljesül, akkor először rendezzük őket):

$$s_1 \leqslant s_2 \leqslant \cdots \leqslant s_n.$$

Feltételünk szerint $s_1 = s$.

A formulák kompakt felírhatóságának érdekében legyen $s_{n+1} = f$ és legyen m az a legkisebb $2 \leqslant j \leqslant n+1$ index, amelyre $s_j > s$. Legyen továbbá minden $m \leqslant i \leqslant n+1$ esetén $\gamma(i)$ az a legnagyobb $j < i$ index, amelyre $s_j < s_i$.

Ezek után minden $m \leqslant i \leqslant n+1$ esetén jelölje $v[i]$ azon feladat egy optimális megoldásában a programokért fizetendő részvételi díjak összegét, amikor a $p_1, p_2, \ldots, p_{\gamma(i)}$ programok közül választhatunk és a fesztivál elejétől az s_i időpontig kell kitölteni az időt.

Legyen most $m \leqslant i \leqslant n+1$ és legyen $\mathcal{O}$ egy optimális megoldása annak a feladatnak, amikor a $p_1, p_2, \ldots, p_{\gamma(i)}$ programok közül választhatunk és a fesztivál elejétől az s_i időpontig kell kitölteni az időt. Világos, hogy ekkor van olyan $\mathcal{O}$-beli p_j program, amelyre $f_j \geqslant s_i$. Ez a program $\mathcal{O}$ optimalitása miatt egyértelmű és az összes többi $\mathcal{O}$-beli program, ha van ilyen, az s_j időpont előtt kezdődik. Ha $s_j = s$, akkor nyilván $\mathcal{O} = \{p_j\}$, a költség pedig

w_j. Ha viszont $s_j > s$, akkor $\mathcal{O}' = \mathcal{O} \setminus \{p_j\}$ szükségképpen egy optimális megoldása annak a feladatnak, amikor a $p_1, p_2, \ldots, p_{\gamma(j)}$ programok közül választhatunk (ezek kezdődnek az s_j időpont előtt), és a fesztivál elejétől az s_j időpontig kell kitölteni az időt. Valóban, ha lenne az utóbbi feladatnak $\mathcal{O}'$-nél kedvezőbb megoldása, akkor ezt a megoldást a p_j programmal kiegészítve egy $\mathcal{O}$-nál kedvezőbb megoldását kapnánk annak a feladatnak, amikor a $p_1, p_2, \ldots, p_{\gamma(i)}$ programok közül választhatunk és a fesztivál elejétől az s_i időpontig kell kitölteni az időt, ellentmondás. A költség ekkor $w_j + v[j]$. Egy probléma van: nem ismerjük j értékét. Azonban j legfeljebb $\gamma(i) \leqslant n$ különböző értéket vehet fel; vizsgáljuk meg az összes lehetőséget, és válasszuk ki a legkedvezőbbet. Így

$$v[i] = \min(\{w_j \mid 1 \leqslant j \leqslant \gamma(i),\ f_j \geqslant s_i \text{ és } s_j = s\} \cup \\ \cup \{w_j + v[j] \mid 1 \leqslant j \leqslant \gamma(i),\ f_j \geqslant s_i \text{ és } s_j > s\}).$$

A $v[i]$ értékek kiszámítását úgy tekinthetjük, mintha egy $n + 2 - m$ elemű tömböt töltenénk ki balról jobbra. A feladat optimális megoldásában a programokért fizetendő részvételi díjak összege $v[n + 1]$.

Minden $v[i]$ érték $O(n)$ lépésben számítható, így a $v[m : n + 1]$ tömb kitöltésének költsége $O(n^2)$. Ehhez jön még a kezdeti rendezés költsége, ami $O(n \log n)$, valamint a $\gamma(i)$ értékek meghatározásának költsége, ami $O(n)$. Így az algoritmus teljes költsége $O(n \log n) + O(n) + O(n^2) = O(n^2)$.

Ha minden $m \leqslant i \leqslant n + 1$ esetén feljegyezzük, hogy $v[i]$ számításánál mely p_j program választása bizonyult a legkedvezőbbnek, akkor magát egy optimális megoldást is könnyen rekonstruálhatunk.

Ha minden program elvi részvételi díja ugyanakkora, akkor a feladat egyszerűen P-beli programok egy olyan minimális elemszámú $\{p_{i_1}, p_{i_2}, \ldots, p_{i_k}\}$ részhalmazának a meghatározása, amelyre

$$[s_{i_1}, f_{i_1}] \cup [s_{i_2}, f_{i_2}] \cup \cdots \cup [s_{i_k}, f_{i_k}] = [s, f].$$

Erre a feladatra a következő mohó algoritmus is optimális megoldást ad. Tegyük fel ismét, hogy a programok a kezdési idejük szerint monoton növekvően rendezettek, vagyis $s_1 \leqslant s_2 \leqslant \cdots \leqslant s_n$.

A kiválasztott programokat egy H halmazban fogjuk tárolni. Kezdetben legyen $H = \emptyset$. Használni fogunk még egy G listát, amely mindig a még nem vizsgált programokat tartalmazza a kezdési idejük szerint monoton növekvően rendezetten. Kezdetben G az összes P-beli programból áll. Először töröljük G-ből azokat a programokat, amelyek kezdési időpontja s, majd ezek közül vegyünk hozzá H-hoz egy legkésőbbi befejezési időpontút. Ismételjük

ezek után a következőt, amíg G-ből nem töröltük az összes programot. Tegyük fel, hogy H-hoz legutóbb az m_j programot vettük hozzá. Ha $f_j = f$, akkor töröljük G-ből az összes programot (ezzel befejeződik az algoritmus). Ha viszont $f_j < f$, akkor töröljük G-ből azokat a programokat, amelyek az f_j időpont előtt vagy éppen az f_j időpontban kezdődnek, majd ezek közül vegyünk hozzá H-hoz egy legkésőbbi befejezési időpontút (jegyezzük meg, hogy ez a program szükségképpen az f_j időpont után fejeződik be).

Legyen $\mathcal{H} = \{p_{j_1}, p_{j_2}, \ldots, p_{j_l}\}$ az algoritmus által kiválasztott programok halmaza, ahol $j_1 < j_2 < \cdots < j_l$. Világos módon

$$[s_{j_1}, f_{j_1}] \cup [s_{j_2}, f_{j_2}] \cup \cdots \cup [s_{j_l}, f_{j_l}] = [s, f].$$

Belátjuk, hogy $\mathcal{H}$ egy optimális megoldása a feladatnak. Tekintsük a feladat egy $\mathcal{O} = \{p_{i_1}, p_{i_2}, \ldots, p_{i_k}\}$ optimális megoldását, ahol $i_1 < i_2 < \cdots < i_k$. Nyilván $l \geqslant k$.

Először teljes indukcióval megmutatjuk, hogy $f_{j_r} \geqslant f_{i_r}$ minden $1 \leqslant r \leqslant k$ esetén. Ha $r = 1$, akkor ez nyilvánvaló, hiszen az algoritmus elsőként az s időpontban kezdődő programok közül egy legnagyobb befejezési időpontút választ ki. Legyen most $r > 1$ és tegyük fel, hogy $f_{j_{r-1}} \geqslant f_{i_{r-1}}$. Mivel $\mathcal{O}$ egy optimális megoldása a feladatnak, ezért $s_{i_r} \leqslant f_{i_{r-1}}$. Másrészt az $f_{j_{r-1}}$ időpont előtt vagy éppen az $f_{j_{r-1}}$ időpontban kezdődő programok közül m_{j_r} egy legkésőbbi befejezési időpontú. Ennélfogva $f_{j_r} \geqslant f_{i_r}$.

Indirekt tegyük fel ezután, hogy $\mathcal{H}$ nem egy optimális megoldása a feladatnak, vagyis $l > k$. Az előzőek szerint $f_{j_k} \geqslant f_{i_k}$. Mivel $\mathcal{O}$ egy optimális megoldása a feladatnak, ezért $f_{i_k} = f$, következésképpen $f_{j_k} = f$ szintén. Ez viszont azt jelenti, hogy az algoritmus az m_{j_k} program kiválasztása után befejeződött, ami ellentmond annak, hogy $l > k$. Ebből következik, hogy $\mathcal{H}$ egy optimális megoldása a feladatnak.

Az algoritmus költsége a kezdeti rendezés nélkül $O(n)$, azzal együtt

$$O(n \log n) + O(n) = O(n \log n).$$

Ütemezés

Egy megbízható vállalkozót sokan keresnek meg különböző munkákkal. Minden munkához tartozik egy elvégzési idő, egy határidő és egy munkadíj. A vállalkozó egyszerre csak egy munkán tud dolgozni és ha egy munkát elkezdett, azt annak befejezéséig nem szakítja meg. Egy munkáért a munkadíj akkor jár, ha az legkésőbb a határidőre elkészül. Mely munkákat vállalja el a vállalkozó és azokat mikor végezze el, ha a bevételét maximalizálni akarja?

Formálisan, adott a munkák $M = \{m_1, m_2, \ldots, m_n\}$ halmaza. Minden m_i munkának adott a t_i végrehajtási ideje és a d_i határideje, ahol t_i és d_i pozitív egész számok (mondjuk napok). Ezen kívül minden m_i munkához adott az érte kapható w_i munkadíj, ahol $w_i > 0$. Feladatunk M-beli munkák egy olyan $\{m_{i_1}, m_{i_2}, \ldots, m_{i_k}\}$ részhalmazának a meghatározása, amelyre a következők teljesülnek. Minden m_{i_j} munkához megadható egy s_{i_j} kezdési és egy f_{i_j} befejezési időpont, ahol s_{i_j} és f_{i_j} pozitív egészek számok (napok), úgy, hogy $f_{i_j} - s_{i_j} + 1 = t_{i_j}$ és $f_{i_j} \leqslant d_{i_j}$, bármely két különböző m_{i_j} és m_{i_h} munkára $f_{i_j} < s_{i_h}$ vagy $f_{i_h} < s_{i_j}$, és ezen feltételek mellett $w_{i_1} + w_{i_2} + \cdots + w_{i_k}$ maximális!

A feladatot dinamikus programozással oldjuk meg. Tegyük fel, hogy a munkák a határidejük szerint monoton növekvően rendezettek (ha ez nem teljesül, akkor először rendezzük őket):

$$d_1 \leqslant d_2 \leqslant \cdots \leqslant d_n.$$

Minden $1 \leqslant i \leqslant n$ és $1 \leqslant j \leqslant d_n$ esetén jelölje $v[i, j]$ azon feladat egy optimális megoldásában az összbevételt, amikor az $m_i, m_{i+1}, \ldots, m_n$ munkák közül választhatunk és az első munkát legkorábban a j-edik nap kezdhetjük el. A formulák kompakt felírhatóságának érdekében legyen $v[i, j] = 0$, ha $i = n + 1$ vagy $j = d_n + 1$.

Legyen most $1 \leqslant i \leqslant n$ és $1 \leqslant j \leqslant d_n$ és legyen $\mathcal{O}$ egy optimális megoldása annak a feladatnak, amikor az $m_i, m_{i+1}, \ldots, m_n$ munkák közül választhatunk és az első munkát legkorábban a j-edik nap kezdhetjük el. Ha $j + t_i - 1 > d_i$, akkor m_i nyilván nem szerepelhet $\mathcal{O}$-ban, így $\mathcal{O}$ szükségképpen egy optimális megoldása annak a feladatnak, amikor az $m_{i+1}, \ldots, m_n$ munkák közül választhatunk és az első munkát legkorábban a j-edik nap kezdhetjük el. Ebben az esetben a bevétel $v[i+1, j]$. Tegyük fel ezután, hogy $j + t_i - 1 \leqslant d_i$.

- Ha m_i szerepel az $\mathcal{O}$-beli munkák között, akkor az az $\mathcal{O}'$ ütemezés, amelyet úgy kapunk $\mathcal{O}$-ból, hogy elhagyjuk az m_i munkát, az m_i elé ütemezett munkákat pedig mind t_i nappal későbbre ütemezzük (a munkáknak a határidők szerinti rendezettsége miatt ezek továbbra is határidőn belül vannak), egy optimális megoldása annak a feladatnak, amikor az $m_{i+1}, \ldots, m_n$ munkák közül választhatunk és az első munkát legkorábban a $(j + t_i)$-edik nap kezdhetjük el. Valóban, ha lenne $\mathcal{O}'$-nél nagyobb bevételt hozó ütemezés arra a feladatra, amikor az $m_{i+1}, \ldots, m_n$ munkák közül választhatunk és az első munkát legkorábban a $(j + t_i)$-edik nap kezdhetjük el, akkor ehhez hozzávéve az m_i munkát j-edik napi kezdéssel, egy $\mathcal{O}$-nál nagyobb bevételt hozó ütemezést kapnánk arra a feladatra, amikor az $m_i, m_{i+1}, \ldots, m_n$ munkák közül választhatunk és

az első munkát legkorábban a j-edik nap kezdhetjük el, ellentmondás. A bevétel ekkor $w_i + v[i+1, j+t_i]$.

- Ha m_i nem szerepel az $\mathcal{O}$-beli munkák között, akkor $\mathcal{O}$ szükségképpen egy optimális megoldása annak a feladatnak, amikor az $m_{i+1}, \ldots, m_n$ munkák közül választhatunk és az első munkát legkorábban a j-edik nap kezdhetjük el. A bevétel ekkor $v[i+1, j]$.

Mivel nem tudjuk, hogy m_i szerepel-e az $\mathcal{O}$-beli munkák között, ezért mindkét lehetőséget megvizsgáljuk és a kedvezőbbet választjuk. Így

$$v[i,j] = \begin{cases} v[i+1,j] & \text{ha } j + t_i - 1 > d_i, \\ \max(w_i + v[i+1, j+t_i], v[i+1,j]) & \text{ha } j + t_i - 1 \leqslant d_i. \end{cases}$$

A $v[i, j]$ értékek kiszámítását úgy tekinthetjük, mintha egy $n+1$ sorból és $d_n + 1$ oszlopból álló táblázatot töltenénk ki alulról felfelé, soronként jobbról balra. A feladat optimális megoldásában az összbevétel $v[1, 1]$.

Minden $v[i, j]$ érték $O(1)$ lépésben számítható, így a v tömb kitöltésének költsége $O(nd_n)$. Ez azt jelenti, hogy az algoritmus nem polinomiális. Vigasztalásul megjegyezzük, hogy a feladatra nem ismert polinomiális algoritmus.

Ha minden $1 \leqslant i \leqslant n$ és $1 \leqslant j \leqslant d_n$ esetén feljegyezzük, hogy $v[i, j]$ számításánál kedvezőbb volt-e az m_i munkát elvállalni vagy nem, akkor magát egy optimális megoldást is könnyen rekonstruálhatunk (a kiválasztott munkákat az első naptól kezdődően, egymás után, szünet nélkül osszuk be).

Ha minden munkáért ugyanannyi a munkadíj, akkor a feladat egyszerűen az adott feltételeknek eleget tevő munkák egy maximális elemszámú halmazának a meghatározása. Erre a feladatra a következő mohó algoritmus is optimális megoldást ad. Tegyük fel ismét, hogy a munkák a határidejük szerint monoton növekvően rendezettek.

Az elvállalt munkákat egy H halmazban fogjuk tárolni. Kezdetben legyen $H = \emptyset$. Most egymás után, minden $1 \leqslant i \leqslant n$ esetén vegyük hozzá az m_i munkát H-hoz. Ha így

$$\sum_{m_j \in H} t_j > d_i$$

adódik, akkor töröljünk egy leghosszabb elvégzési idejű munkát H-ból. Végül a H-beli munkákat az első naptól kezdve, szünet nélkül, a határidejük szerint monoton növekvő sorrendbe osszuk be. Jelölje $\mathcal{H}$ a munkák ilyen módon történő ütemezését. Jegyezzük meg, hogy a $\mathcal{H}$ ütemezésben minden munka legkésőbb a határidejére elkészül.

A munkák n száma szerinti teljes indukcióval bizonyítjuk, hogy $\mathcal{H}$ egy optimális megoldása a feladatnak. Ha $n = 1$, akkor ez nyilvánvaló. Legyen ezután $n > 1$, és tegyük fel, hogy $n - 1$ munka esetén igaz az állítás.

Ha $\mathcal{H}$ az összes munkát tartalmazza, akkor optimális megoldása a feladatnak világos módon. Tegyük fel ezért, hogy van olyan munka, amely nem szerepel $\mathcal{H}$-ban. Legyen ezek közül m_k az a munka, amelyet először törölt az algoritmus H-ból, mondjuk az m_i munka hozzávételekor.

Megmutatjuk, hogy a feladatnak van olyan $\mathcal{O}$ optimális megoldása, amelyben a munkák az első naptól kezdve, szünet nélkül, a határidejük szerint monoton növekvő sorrendben követik egymást, és amelyben szintén nem szerepel az m_k munka. Az világos, hogy a feladatnak létezik olyan optimális megoldása, amelyben a munkák az első naptól kezdve, szünet nélkül, a határidejük szerint monoton növekvő sorrendben követik egymást. Tegyük fel, hogy egy ilyen $\mathcal{O}'$ optimális megoldásban szerepel az m_k munka. Mivel

$$\sum_{j=1}^{i} t_j > d_i,$$

ezért az $m_1, m_2, \ldots, m_i$ munkák között szükségképpen van olyan m_l munka, amely nem szerepel $\mathcal{O}'$-ben. Legyen $\mathcal{O}$ az az ütemezés, amelyet úgy kapunk $\mathcal{O}'$-ből, hogy az m_k munkát lecseréljük az m_l munkára, majd a munkákat az első naptól kezdve, szünet nélkül, a határidejük szerint monoton növekvő sorrendbe osszuk be. Most az $m_1, m_2, \ldots, m_i$ közül való $\mathcal{O}$-beli munkák legkésőbb a határidejükre mind elkészülnek, hiszen i megválasztása miatt ugyanez akkor is teljesül, ha az összes $m_1, \ldots, m_{k-1}, m_{k+1}, \ldots, m_i$ munkát osztjuk be az első naptól kezdve, szünet nélkül. Másrészt $t_k \geqslant t_l$ miatt az $m_{i+1}, m_{i+2}, \ldots, m_n$ közül való $\mathcal{O}$-beli munkák szintén legkésőbb a határidejükre mind elkészülnek, hiszen semelyik nincs későbbre ütemezve $\mathcal{O}$-ban, mint amikorra $\mathcal{O}'$-ben volt. Ebből következik, hogy $\mathcal{O}$ is egy optimális megoldása a feladatnak.

Nyilván a $\mathcal{H}$-beli munkák száma nem haladhatja meg az $\mathcal{O}$-beli munkák számát, hiszen az utóbbi ütemezés optimális megoldása a feladatnak. Másrészt a mohó algoritmust a munkák $M \setminus \{m_k\}$ halmazára futtatva szintén a $\mathcal{H}$ ütemezést kapjuk, ami az indukciós feltevés szerint optimális a munkák $M \setminus \{m_k\}$ halmazára. Mivel az $\mathcal{O}$ ütemezésben is csak $M \setminus \{m_k\}$ halmazbeli munkák vannak, így az $\mathcal{O}$-beli munkák száma sem haladhatja meg a $\mathcal{H}$-beli munkák számát. Ebből következik, hogy a $\mathcal{H}$-beli munkák száma megegyezik az $\mathcal{O}$-beli munkák számával, következésképpen $\mathcal{H}$ is optimális megoldása az eredeti feladatnak.

A H-beli munkákat az elvégzési idők szerint rendezett maximum kupacban tárolva a soron következő munka beszúrásának valamint egy maximális elvégzési idejű munka esetleges törlésének költsége $O(\log n)$. A törlés szükségessége $O(1)$ lépésben eldönthető, ha egy segédváltozóban nyilvántartjuk a kupacban levő munkák elvégzési idejének összegét. Ezt minden $1 \leqslant i \leqslant n$ esetén végrehajtva az összköltség $O(n \log n)$. Ehhez jön még a kezdeti rendezés $O(n \log n)$ költsége, így az algoritmus teljes költsége $O(n \log n)$.

Ha minden munka végrehajtási ideje 1 nap, akkor a következő mohó algoritmus is optimális megoldást ad. Tegyük fel most, hogy a munkák a munkadíjuk szerint monoton csökkenően rendezettek (ha ez nem teljesül, akkor először rendezzük őket):

$$w_1 \geqslant w_2 \geqslant \cdots \geqslant w_n.$$

Most egymás után minden $1 \leqslant i \leqslant n$ esetén ütemezzük be az m_i munkát a d_i-edik illetve az azt megelőző napok közül a legkésőbbi olyanra, amely még szabad. Ha az összes ilyen nap foglalt már, akkor a munkát ne vállaljuk el. Jelölje $\mathcal{H}$ a munkák ilyen módon történő ütemezését.

Először belátjuk, hogy $\mathcal{H}$ egy optimális megoldása a feladatnak. Tekintsük a munkák egy optimális $\mathcal{O}$ ütemezését. Ha $\mathcal{O} = \mathcal{H}$, akkor készen vagyunk. Tegyük fel ezért, hogy $\mathcal{O} \neq \mathcal{H}$. Legyen k a legkisebb olyan index, amelyre m_k ütemezése különbözik $\mathcal{O}$-ban és $\mathcal{H}$-ban. Ekkor minden $j < k$ esetén az m_j munka

- vagy nem szerepel sem $\mathcal{O}$-ban sem $\mathcal{H}$-ban,
- vagy szerepel $\mathcal{O}$-ban és $\mathcal{H}$-ban is, mindkétszer ugyanarra a napra ütemezve.

Négy esetet különböztethetünk meg.

(1) Az m_k munka szerepel $\mathcal{O}$-ban, de nem szerepel $\mathcal{H}$-ban. Mivel m_k szerepel $\mathcal{O}$-ban, ezért a $\mathcal{H}$-t előállító algoritmusunk k-adik lépésében a d_k-adik illetve az azt megelőző napok között szükségképpen van még szabad nap. Ám ez ellentmond annak, hogy $\mathcal{H}$-ból kihagytuk m_k-t.

(2) Az m_k munka szerepel $\mathcal{H}$-ban, de nem szerepel $\mathcal{O}$-ban. Ha arra a napra, amelyre $\mathcal{H}$ ütemezi m_k-t $\mathcal{O}$ nem ütemez egyetlen munkát sem, akkor m_k-t vegyük hozzá $\mathcal{O}$-hoz erre a napra ütemezve. Ezzel a bevétel nyilván nem csökken. Ha viszont arra a napra, amelyre $\mathcal{H}$ ütemezi m_k-t $\mathcal{O}$ is ütemez egy m_j munkát, akkor $\mathcal{O}$-ban cseréljük le m_j-t m_k-ra. Mivel $j > k$, ezért $w_j \leqslant w_k$, következésképpen a bevétel most sem csökken.

(3) Az m_k munka szerepel $\mathcal{O}$-ban és $\mathcal{H}$-ban is, de az előbbiben korábbi napra van ütemezve, mint az utóbbiban. Ha arra a napra, amelyre $\mathcal{H}$ ütemezi m_k-t $\mathcal{O}$ nem ütemez egyetlen munkát sem, akkor m_k-t ütemezzük át $\mathcal{O}$-ban erre a napra. Ezzel a bevétel nem változik. Ha viszont arra a napra, amelyre $\mathcal{H}$ ütemezi m_k-t $\mathcal{O}$ is ütemez egy m_j munkát, akkor $\mathcal{O}$-ban cseréljük fel m_j-t és m_k-t. Mivel m_j így előbbre kerül, a csere megengedett. A bevétel nyilván most sem változik.

(4) Az m_k munka szerepel $\mathcal{O}$-ban és $\mathcal{H}$-ban is, de az utóbbiban korábbi napra van ütemezve, mint az előbbiben. Ez ellentmond annak, hogy a $\mathcal{H}$-t előállító algoritmusunk a k-adik lépésben az m_k munkát a d_k-adik illetve az azt megelőző napok között a legkésőbbi, még szabad napra ütemezte.

Ezzel beláttuk, hogy az optimális $\mathcal{O}$ ütemezés áttranszformálható egy olyan optimális $\mathcal{O}'$ ütemezéssé, hogy $\mathcal{O}'$-ben és $\mathcal{H}$-ban már m_k ütemezése is megegyezik. Az eljárást folytatva $\mathcal{O}$ lépésről-lépésre áttranszformálható $\mathcal{H}$-ba a bevétel csökkenése nélkül. Következésképpen $\mathcal{H}$ is egy optimális ütemezés.

Az algoritmus költsége nagyban függ attól, hogy a k-adik lépésben hogyan találjuk meg a d_k-adik illetve az azt megelőző napok közül a legkésőbbi olyat, amely még szabad. Kézenfekvő ötlet, hogy a d_k-adik naptól egyesével haladjunk visszafelé, ekkor a költség nyilván $O(n)$. Ezt minden $1 \leqslant k \leqslant n$ esetén végrehajtva az összköltség $O(n^2)$. Ehhez jön még a kezdeti rendezés $O(n \log n)$ költsége, így az algoritmus teljes költsége $O(n^2)$.

Megjegyezzük, hogy a szakasz elején ismertetett dinamikus programozás algoritmussal is meghatározhatjuk a feladat egy optimális megoldását $O(n^2)$ lépésben. Ehhez elég észrevenni, hogy a feladatnak mindig van olyan optimális megoldása, amelyben az összes elvállalt munkát legkésőbb az n-edik nap elvégezzük, hiszen az üres napokat meg tudjuk szüntetni a feltételek megsértése nélkül a munkák alkalmas előbbre ütemezésével. Ebből következik, hogy ha minden olyan munkának a határidejét az n-edik napra hozzuk előre, amelyiknek a határideje az n-edik nap után van, akkor az így módosított feladat egy optimális megoldásában az összbevétel nyilván ugyanannyi lesz, mint az eredeti feladat esetén. Erre a módosított feladatra futtatva a dinamikus programozás algoritmust, a fenti költség adódik.

Az UNIÓ-HOLVAN adatszerkezet használatával lényegesen hatékonyabban is megvalósítható a mohó algoritmus. Először is hozzuk előre az n-edik napra minden olyan munkának a határidejét, amelyiknek a határideje az n-edik nap után van. Tartsuk ezután nyilván az egymás utáni foglalt napok maximális halmazait (amelyeket a közvetlenül előttük lévő szabad nappal azonosítunk), valamint a szabad napokat, mint egyelemű halmazokat. Az algoritmus k-adik lépésében ellenőrizzük, hogy a d_k-adik nap szabad-e, illetve

ha nem, akkor a foglalt napok mely blokkjához tartozik. Az első esetben m_k-t a d_k-adik napra ütemezzük, a másodikban a blokkot közvetlenül megelőző szabad napra (illetve eldobjuk, ha nincs a blokk előtt szabad nap). Ha az a nap, amelyre m_k-t ütemeztük, foglalt napok blokkjaival szomszédos, akkor képezzük a blokkok unióját. A költség így csupán $O(\log n)$. Az algoritmus ezen implementációjának teljes költsége ennélfogva $O(n \log n)$.

Végül jegyezzük meg, hogy ha minden munkának ugyanaz a határideje, jelölje ezt mondjuk D, akkor a feladat az M-beli munkák egy olyan $\{m_{i_1}, m_{i_2}, \ldots, m_{i_k}\}$ részhalmazának a meghatározása, amelyre $t_{i_1} + t_{i_2} + \cdots + t_{i_k} \leqslant D$ és $w_{i_1} + w_{i_2} + \cdots + w_{i_k}$ maximális. Ez a jól ismert hátizsák probléma, amelyre szintén nem ismert polinomiális algoritmus.

Visszajáró pénz

A visszajáró pénz problémánál adott pénzmennyiséget kell kifizetni a lehető legkevesebb érme felhasználásával. Legyen a kifizetendő pénzmennyiség $n \in \mathbb{N}$ és legyenek a felhasználható címletek $c_1, c_2, \ldots, c_m$, ahol

$$c_1 > c_2 > \cdots > c_m = 1$$

pozitív egész számok. Tegyük fel, hogy minden címletből tetszőlegesen sok érme áll a rendelkezésünkre. Ekkor bármely pénzmennyiség kifizethető, hiszen a címletek között az 1 is szerepel.

A feladatot dinamikus programozással oldjuk meg. Minden $0 \leqslant j \leqslant n$ esetén jelölje $v[j]$ a kifizetésben szereplő érmék minimális számát, amikor a kifizetendő pénzmennyiség j. Ha $j = 0$, akkor $v[j] = 0$ nyilvánvaló módon. Legyen ezután $1 \leqslant j \leqslant n$. Tekintsük most egy legkevesebb érmét felhasználó megoldását annak a feladatnak, amikor a kifizetendő pénzmennyiség j. Legyen a megoldásban felhasznált érmék száma s. Ha most ezek közül elhagyunk egyet, mondjuk egy c_i címletűt, akkor a megmaradt $s - 1$ érme egy legkevesebb érmét felhasználó megoldását adja annak a feladatnak, amikor a kifizetendő pénzmennyiség $j - c_i$. Valóban, ha ez utóbbi pénzmennyiség kifizethető lenne $(s-1)$-nél kevesebb érmével is, akkor ezeket az érméket az elhagyott c_i címletű érmével kiegészítve az eredeti feladat egy s-nél kevesebb érmét felhasználó megoldásához jutnánk, ami ellentmondás. Ebből következik, hogy esetünkben $v[j] = 1 + v[j - c_i]$. Ez a rekurzív egyenlet feltételezi, hogy ismerjük i értékét, miközben ez nem igaz. Azonban i legfeljebb m különböző értéket vehet fel; vizsgáljuk meg az összes lehetőséget, és válasszuk ki a legkedvezőbbet. Így

$$v[j] = 1 + \min\{v[j - c_i] \mid 1 \leqslant i \leqslant m,\ c_i \leqslant j\}.$$

A $v[j]$ értékek kiszámítását úgy tekinthetjük, mintha egy $n+1$ elemű tömböt töltenénk ki balról jobbra. A feladat optimális megoldásában az érmék száma $v[n]$.

Minden $v[j]$ érték $O(m)$ lépésben számítható, így a $v[0:n]$ tömb kitöltésének költsége $O(mn)$. Ez azt jelenti, hogy az algoritmus nem polinomiális. Vigasztalásul megjegyezzük, hogy a feladatra nem ismert polinomiális algoritmus.

Ha minden $1 \leqslant j \leqslant n$ esetén feljegyzünk egy olyan c_i címletet is, amelyre $v[j] = 1 + v[j - c_i]$, akkor magát egy legkevesebb érméből álló kifizetést is könnyen rekonstruálhatunk.

A legtöbb ismert pénzrendszer esetén a következő mohó algoritmus is optimális megoldást ad: amíg $n > 0$, addig keressük meg azt a legnagyobb c_i címletet, amely kisebb vagy egyenlő, mint n, adjunk ki egy c_i címletű érmét, majd oldjuk meg rekurzívan azt a feladatot, amikor a kifizetendő pénzmennyiség $n - c_i$. Az algoritmus költsége $O(m)$; egymás után egy-egy osztással meghatározhatjuk a szükséges c_i címletű érmék számát minden $1 \leqslant i \leqslant m$ esetén.

Vannak azonban kivételek. Az 1971 előtti angol pénzrendszerben a címletek penny-ben kifejezve a következők voltak: 1, 3, 6, 12, 24, 30, 60, 240 (igazság szerint forgalomban volt még félpenny is, de ez nem játszik szerepet a példánkban). Itt a mohó algoritmussal 48 penny-t 3 érmével fizetnénk ki: $30 + 12 + 6$. Ezzel szemben az optimális megoldás csak két érméből áll: $24 + 24$.

Melyek azok a pénzrendszerek, amelyekre működik a mohó algoritmus és melyek azok, amelyekre nem? Ez a kérdés meglepően hatékony módon eldönthető. Az algoritmus ismertetése előtt némi előkészület szükséges.

A címletek monoton csökkenő sorozatára vezessük be a

$$\boldsymbol{c} = (c_1, c_2, \ldots, c_m)$$

jelölést. Adott $n \in \mathbb{N}$ esetén egy

$$\boldsymbol{r} = (r_1, r_2, \ldots, r_m) \in \mathbb{N}^m$$

sorozatot az n egy $\boldsymbol{c}$-beli reprezentációjának fogunk nevezni, ha

$$\boldsymbol{rc} = r_1 c_1 + r_2 c_2 + \cdots + r_m c_m = n.$$

Vezessük még be az $|\boldsymbol{r}| = r_1 + r_2 + \cdots + r_m$ jelölést, erre a mennyiségre az $\boldsymbol{r}$ reprezentáció méreteként fogunk hivatkozni. A visszajáró pénz feladat ebben

a megfogalmazásban adott $n \in \mathbb{N}$ egy minimális méretű $\boldsymbol{c}$-beli reprezentációjának a megtalálása.

Definiáljuk az $\mathbb{N}^m$-beli sorozatok halmazán a $\preccurlyeq_1$, $\preccurlyeq_2$, $\preccurlyeq_3$ rendezési relációkat a következőképpen. Legyenek $\boldsymbol{x} = (x_1, x_2, \ldots, x_m) \in \mathbb{N}^m$ és $\boldsymbol{y} = (y_1, y_2, \ldots, y_m) \in \mathbb{N}^m$. Azt fogjuk mondani, hogy

- $\boldsymbol{x} \preccurlyeq_1 \boldsymbol{y}$, ha vagy $\boldsymbol{x} = \boldsymbol{y}$ vagy pedig $x_i < y_i$ a legkisebb olyan $1 \leqslant i \leqslant m$ indexre, amelyre $x_i \neq y_i$,
- $\boldsymbol{x} \preccurlyeq_2 \boldsymbol{y}$, ha vagy $|\boldsymbol{x}| > |\boldsymbol{y}|$ vagy pedig $|\boldsymbol{x}| = |\boldsymbol{y}|$ és $\boldsymbol{x} \preccurlyeq_1 \boldsymbol{y}$,
- $\boldsymbol{x} \preccurlyeq_3 \boldsymbol{y}$, ha $x_i \leqslant y_i$ minden $1 \leqslant i \leqslant m$ esetén.

Megjegyezzük, hogy tetszőleges $\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z} \in \mathbb{N}^m$ sorozatokra

$$\boldsymbol{x} \preccurlyeq_k \boldsymbol{y} \iff \boldsymbol{x} + \boldsymbol{z} \preccurlyeq_k \boldsymbol{y} + \boldsymbol{z}$$

mindhárom rendezési reláció esetén.

A továbbiakban, ha ez nem okoz félreértést, a $\boldsymbol{c}$-beli reprezentáció kifejezés elejéről el fogjuk hagyni a $\boldsymbol{c}$-beli jelzőt. Adott n esetén jelölje $\boldsymbol{g}(n)$ a $\preccurlyeq_1$ rendezés szerint maximális reprezentációját n-nek. A $\boldsymbol{g}(n)$ sorozatra az n mohó reprezentációjaként fogunk hivatkozni. Vegyük észre, hogy ha $\boldsymbol{g}(n) = (g_1, g_2, \ldots, g_m)$, akkor a visszajáró pénz problémára adott mohó algoritmus éppen g_i darab c_i címletű érmét használ n kifizetéséhez minden $1 \leqslant i \leqslant m$ esetén. Jelölje továbbá adott n esetén $\boldsymbol{h}(n)$ a $\preccurlyeq_2$ rendezés szerint maximális reprezentációját n-nek. A $\boldsymbol{h}(n)$ sorozatra az n minimális reprezentációjaként fogunk hivatkozni. Jegyezzük meg, hogy $\boldsymbol{h}(n)$ az n minimális méretű reprezentációi közül a $\preccurlyeq_1$ rendezés szerint maximális.

Azt fogjuk mondani, hogy egy $\boldsymbol{c}$ pénzrendszer kanonikus, ha $\boldsymbol{g}(n) = \boldsymbol{h}(n)$ minden $n \in \mathbb{N}$ esetén. Ez azt jelenti, hogy a visszajáró pénz problémára adott mohó algoritmus az adott $\boldsymbol{c}$ pénzrendszert használva mindig optimális megoldást ad. Célunk ebben a megfogalmazásban a kanonikus pénzrendszerek azonosítása.

1. Állítás. Legyenek $n', n'' \in \mathbb{N}$. Ha $n' < n''$, akkor $\boldsymbol{g}(n') \prec_1 \boldsymbol{g}(n'')$.

Bizonyítás. Ha $n' < n''$, akkor $\boldsymbol{r} = \boldsymbol{g}(n') + (0, 0, \ldots, 0, n'' - n')$ egy reprezentációja n''-nek. Most egyrészt $\boldsymbol{g}(n') \prec_1 \boldsymbol{r}$ nyilvánvaló módon, másrészt $\boldsymbol{r} \preccurlyeq_1 \boldsymbol{g}(n'')$ a mohó reprezentáció definíciójával összhangban. Így $\boldsymbol{g}(n') \prec_1 \boldsymbol{g}(n'')$.

2. Állítás. Legyenek $n', n'' \in \mathbb{N}$, és legyen $\boldsymbol{r'}$ egy reprezentációja n'-nek, valamint $\boldsymbol{r''}$ egy reprezentációja n''-nek. Tegyük fel, hogy $\boldsymbol{r'} \preccurlyeq_3 \boldsymbol{r''}$.

(1) Ha $\boldsymbol{r''}$ mohó reprezentációja n''-nek, akkor $\boldsymbol{r'}$ is mohó reprezentációja n'-nek.

(2) Ha $\boldsymbol{r''}$ minimális reprezentációja n''-nek, akkor $\boldsymbol{r'}$ is minimális reprezentációja n'-nek.

Bizonyítás. Egy általános észrevétellel kezdjük a bizonyítást. Legyen $\boldsymbol{r}$ egy tetszőleges reprezentációja n'-nek. Ekkor $\boldsymbol{rc} = \boldsymbol{r'c}$. Innen $\boldsymbol{rc} + \boldsymbol{r''c} = \boldsymbol{r'c} + \boldsymbol{r''c}$, illetve átrendezve $\boldsymbol{r''c} - \boldsymbol{r'c} + \boldsymbol{rc} = \boldsymbol{r''c}$, majd kiemelve $(\boldsymbol{r''} - \boldsymbol{r'} + \boldsymbol{r})\boldsymbol{c} = \boldsymbol{r''c}$. Feltételünk szerint $\boldsymbol{r'} \preccurlyeq_3 \boldsymbol{r''}$, így $\boldsymbol{r''} - \boldsymbol{r'} + \boldsymbol{r} \in \mathbb{N}^m$, következésképpen $\boldsymbol{r''} - \boldsymbol{r'} + \boldsymbol{r}$ is egy reprezentációja n''-nek. Lássuk ezután a két állítás nagyon hasonló bizonyítását.

(1) Mivel $\boldsymbol{r''}$ mohó reprezentációja n''-nek, ezért $\boldsymbol{r''} - \boldsymbol{r'} + \boldsymbol{r} \preccurlyeq_1 \boldsymbol{r''}$. Ebből következik, hogy $\boldsymbol{r''} + \boldsymbol{r} \preccurlyeq_1 \boldsymbol{r''} + \boldsymbol{r'}$, ahonnan $\boldsymbol{r} \preccurlyeq_1 \boldsymbol{r'}$ adódik. Ez viszont csak akkor lehetséges n' teteszőleges $\boldsymbol{r}$ reprezentációjára, ha $\boldsymbol{r'}$ mohó reprezentációja n'-nek.

(2) Mivel $\boldsymbol{r''}$ minimális reprezentációja n''-nek, ezért $\boldsymbol{r''} - \boldsymbol{r'} + \boldsymbol{r} \preccurlyeq_2 \boldsymbol{r''}$. Ebből következik, hogy $\boldsymbol{r''} + \boldsymbol{r} \preccurlyeq_2 \boldsymbol{r''} + \boldsymbol{r'}$, ahonnan $\boldsymbol{r} \preccurlyeq_2 \boldsymbol{r'}$ adódik. Ez viszont csak akkor lehetséges n' teteszőleges $\boldsymbol{r}$ reprezentációjára, ha $\boldsymbol{r'}$ minimális reprezentációja n'-nek.

Tegyük fel ezek után, hogy egy $\boldsymbol{c}$ pénzrendszer nem kanonikus. Legyen n a legkisebb olyan természetes szám, amelyre $\boldsymbol{g}(n) \neq \boldsymbol{h}(n)$. Legyen $\boldsymbol{g}(n) = (g_1, g_2, \ldots, g_m)$ és $\boldsymbol{h}(n) = (h_1, h_2, \ldots, h_m)$. Megmutatjuk, hogy minden $1 \leqslant i \leqslant m$ esetén g_i és h_i közül legalább az egyik nulla. Indirekt tegyük fel, hogy ez nem teljesül, azaz valamely $1 \leqslant i \leqslant m$ esetén g_i és h_i is pozitív. Ekkor

$$\boldsymbol{r'} = (g_1, \ldots, g_{i-1}, g_i - 1, g_{i+1}, \ldots, g_m) \in \mathbb{N}^m$$

és

$$\boldsymbol{r''} = (h_1, \ldots, h_{i-1}, h_i - 1, h_{i+1}, \ldots, h_m) \in \mathbb{N}^m$$

az n-nél kisebb $n - c_i \in \mathbb{N}$ szám olyan reprezentációi, amelyekre $\boldsymbol{r'} \prec_3 \boldsymbol{g}(n)$ és $\boldsymbol{r''} \prec_3 \boldsymbol{h}(n)$, így a 2. Állítás szerint $\boldsymbol{r'} = \boldsymbol{g}(n - c_i)$ és $\boldsymbol{r''} = \boldsymbol{h}(n - c_i)$. Ez viszont azt jelenti, hogy $\boldsymbol{g}(n - c_i) \neq \boldsymbol{h}(n - c_i)$, ellentmondva n minimalitásának.

Az n szám minimális reprezentációjában legyen h_α az első és h_β az utolsó pozitív tag. Ekkor az előzőekkel összhangban az n szám mohó reprezentációjában $g_\alpha = 0$. Ám $\boldsymbol{h}(n) \prec_1 \boldsymbol{g}(n)$ miatt ekkor g_i szükségképpen pozitív valamely $i < \alpha$ esetén. Ebből rögtön adódik, hogy $\alpha > 1$.

Meglepő kapcsolat áll fenn n minimális és $c_{\alpha-1}-1$ mohó reprezentációja között. Legyen $\boldsymbol{g}(c_{\alpha-1}-1)=(f_1,f_2,\ldots,f_m)$. Ekkor

$$h_i=\begin{cases} f_i & \text{ha } i<\beta,\\ f_i+1 & \text{ha } i=\beta,\\ 0 & \text{ha } i>\beta. \end{cases}$$

Ez a következőképpen látható be. Mivel g_i pozitív valamely $i<\alpha$ esetén, ezért $n\geqslant c_{\alpha-1}$. Másrészt a 2. Állítással összhangban

$$(h_1,\ldots,h_{\beta-1},h_\beta-1,h_{\beta+1},\ldots,h_m)=\boldsymbol{h}(n-c_\beta).$$

Ugyanakkor $\boldsymbol{h}(n-c_\beta)=\boldsymbol{g}(n-c_\beta)$, hiszen $n-c_\beta<n$ és feltételünk szerint n a legkisebb olyan természetes szám, amelynek minimális és mohó reprezentációja különböző. Felhasználva, hogy $h_i=0$ minden $i<\alpha$ esetén, ebből $n-c_\beta<c_{\alpha-1}$ és így $n-c_\beta\leqslant c_{\alpha-1}-1$ azonnal adódik. Ennélfogva az 1. Állítás szerint

$$\boldsymbol{g}(n-c_\beta)\preccurlyeq_1\boldsymbol{g}(c_{\alpha-1}-1).$$

Tekintsük most a $c_{\alpha-1}-1$ számot. Mivel $c_{\alpha-1}-1\geqslant c_\alpha$, ezért $c_{\alpha-1}-1$ mohó reprezentációjában $f_\alpha\geqslant 1$. Ismét a 2. Állítással összhangban

$$(f_1,\ldots,f_{\alpha-1},f_\alpha-1,f_{\alpha+1},\ldots,f_m)=\boldsymbol{g}(c_{\alpha-1}-1-c_\alpha).$$

Másrészt

$$(h_1,\ldots,h_{\alpha-1},h_\alpha-1,h_{\alpha+1},\ldots,h_m)=\boldsymbol{h}(n-c_\alpha)$$

szintén a 2. Állítás szerint. Ennélfogva

$$(h_1,\ldots,h_{\alpha-1},h_\alpha-1,h_{\alpha+1},\ldots,h_m)=\boldsymbol{g}(n-c_\alpha),$$

hiszen $n-c_\alpha<n$ és feltételünk szerint n a legkisebb olyan természetes szám, amelynek minimális és mohó reprezentációja különböző. Idézzük fel, hogy $c_{\alpha-1}\leqslant n$, következésképpen $c_{\alpha-1}-1-c_\alpha<n-c_\alpha$, így az 1. Állítás szerint $\boldsymbol{g}(c_{\alpha-1}-1-c_\alpha)\prec_1\boldsymbol{g}(n-c_\alpha)$. Mindkét sorozat α-adik tagját eggyel megnövelve, az előbbiek figyelembe vételével,

$$\boldsymbol{g}(c_{\alpha-1}-1)\prec_1\boldsymbol{h}(n)$$

adódik.

Végül tekintsük a

$$\boldsymbol{g}(n-c_\beta)\preccurlyeq_1\boldsymbol{g}(c_{\alpha-1}-1)\prec_1\boldsymbol{h}(n)$$

láncot. Mivel a $\boldsymbol{g}(n-c_\beta)$ és a $\boldsymbol{h}(n)$ sorozatok csak a β-adik tagban különböznek, ezért $f_i=h_i$ minden $i<\beta$ esetén. Ugyanakkor $\boldsymbol{g}(c_{\alpha-1}-1)\prec_1\boldsymbol{h}(n)$

miatt $f_j < h_j$ valamely $j \geqslant \beta$ indexre. Ám $h_i = 0$ minden $i > \beta$ esetén, ezért $j = \beta$ lehetséges csak, ennélfogva $f_\beta < h_\beta$. Másrészt $\boldsymbol{g}(n - c_\beta) \preccurlyeq_1 \boldsymbol{g}(c_{\alpha-1} - 1)$ miatt $h_\beta - 1 \leqslant f_\beta$, így $h_\beta - 1 \leqslant f_\beta < h_\beta$, ahonnan $h_\beta = f_\beta + 1$ következik.

Ezek után egy $\boldsymbol{c}$ pénzrendszerről már könnyen eldönthető, hogy kanonikus vagy nem. Minden $2 \leqslant \alpha \leqslant \beta \leqslant m$ esetén állítsuk elő a $\boldsymbol{g}(c_{\alpha-1} - 1) = (f_1, f_2, \ldots, f_m)$ sorozatot a mohó algoritmussal, illetve az $\boldsymbol{r} = (r_1, r_2, \ldots, r_m)$ sorozatot, ahol

$$r_i = \begin{cases} f_i & \text{ha } i < \beta, \\ f_i + 1 & \text{ha } i = \beta, \\ 0 & \text{ha } i > \beta. \end{cases}$$

Ha $\boldsymbol{c}$ nem kanonikus, akkor a fentiek szerint az így kapott $\boldsymbol{r}$ sorozatok között találni fogunk olyat, amelyre $|\boldsymbol{g}(\boldsymbol{rc})| > |\boldsymbol{r}|$. Mivel a szóba jövő $\boldsymbol{r}$ sorozatok száma $O(m^2)$, és minden sorozatról $O(m)$ lépésben eldönthető ennek a tulajdonság teljesülése, ezért az algoritmus teljes lépésszáma $O(m^3)$.

Az algoritmus alkalmazásával egyszerűen igazolható, hogy tetszőleges $m \geqslant 2$ és $q \geqslant 2$ egészekre kanonikus az a $\boldsymbol{c} = (c_1, c_2, \ldots, c_m)$ pénzrendszer, ahol $c_i = q^{m-i}$ minden $1 \leqslant i \leqslant m$ esetén. Legyen $2 \leqslant \alpha \leqslant m$ tetszőleges és tekintsük a $\boldsymbol{g}(c_{\alpha-1} - 1) = (f_1, f_2, \ldots, f_m)$ sorozatot. Világos módon

$$f_i = \begin{cases} 0 & \text{ha } i < \alpha, \\ q - 1 & \text{ha } i \geqslant \alpha. \end{cases}$$

Legyen ezután $\alpha \leqslant \beta \leqslant m$ és tekintsük azt az $\boldsymbol{r} = (r_1, r_2, \ldots, r_m)$ sorozatot, ahol

$$r_i = \begin{cases} f_i & \text{ha } i < \beta, \\ f_i + 1 & \text{ha } i = \beta, \\ 0 & \text{ha } i > \beta, \end{cases}$$

vagyis

$$r_i = \begin{cases} 0 & \text{ha } i < \alpha, \\ q - 1 & \text{ha } \alpha \leqslant i < \beta, \\ q & \text{ha } i = \beta, \\ 0 & \text{ha } i > \beta. \end{cases}$$

Vegyük most észre, hogy $\boldsymbol{r}$ éppen a $c_{\alpha-1} = q^{m-\alpha+1}$ szám egy reprezentációja. Valóban

$$\begin{aligned} \boldsymbol{rc} &= (q-1)q^{m-\alpha} + (q-1)q^{m-\alpha-1} + \cdots + (q-1)q^{m-\beta+1} + qq^{m-\beta} \\ &= (q-1)q^{m-\alpha} + (q-1)q^{m-\alpha-1} + \cdots + (q-1)q^{m-\beta+1} + q^{m-\beta+1} \\ &= (q-1)q^{m-\alpha} + (q-1)q^{m-\alpha-1} + \cdots + qq^{m-\beta+1} \\ &= (q-1)q^{m-\alpha} + (q-1)q^{m-\alpha-1} + \cdots + q^{m-\beta+2} \\ &\vdots \end{aligned}$$

$$
\begin{aligned}
&\vdots \\
&= (q-1)q^{m-\alpha} + qq^{m-\alpha-1} \\
&= (q-1)q^{m-\alpha} + q^{m-\alpha} \\
&= qq^{m-\alpha} \\
&= q^{m-\alpha+1}.
\end{aligned}
$$

De $|\boldsymbol{g}(c_{\alpha-1})| = 1$, így szükségképpen $|\boldsymbol{r}| \geqslant |\boldsymbol{g}(c_{\alpha-1})|$. Mivel ez tetszőleges $2 \leqslant \alpha \leqslant \beta \leqslant m$ esetén fennáll, ezért a vizsgált $\boldsymbol{c}$ pénzrendszer kanonikus.