S. 65.

Marussy Kristóf 12. évfolyam
Budapest, Szent István Gimnázium
E-mail: `kris7topher@gmail.com`

1. Az algoritmus formális leírása

1.1. Az aztékok formális nyelve

Tekintsük a $\Sigma = \{<, >\}$ ábécét, és felette az $L \subseteq \Sigma^*$ formális nyelvet, melyre

1. ha $a \in L$, akkor a azonos számú $<$ és $>$ jelet tartalmaz;
2. ha $a \in L$ és $b \sqsubseteq a$ prefixe, akkor b legalább annyi $<$ jelet tartalmaz, mint $>$ jelet,
3. az e üres szó nem eleme L-nek.

A megalkotott L nyelv az Ometeotl szabályai által megengedett jelsorozatokból áll. Az egyszerűség kedvéért nevezzük L elemeit *jó* szavaknak.

1. lemma. *Az jó szavak mind páros sok jelből állnak.*

Bizonyítás. Az állítás a kiegyensúlyozottsági feltétel következménye: ha egy szó k darab $<$ jelet tartalmaz, akkor k darab $>$ jelet kell tartalmazzon, tehát összesen $2k$ darab jelből áll. □

2. lemma. *Bármely jó szónak $<$ prefixe.*

Bizonyítás. A bizonyítást indirekt végezzük.

Tegyük fel, hogy egy a jó szónak $>$ prefixe. Ekkor a-nak van olyan prefixe, mely több $>$ jelet tartalmaz, mint $<$ jelet. Ez ellentmondás, mert sérülne Ometeotl 2. feltétele.

Tehát egyetlen jó szónak sem lehet $>$ prefixe. Mivel e nem jó szó és Σ csak két jelből áll, minden jó szónak $<$ prefixe. □

Az egyszerűség kedvéért azt, hogy a feladat R végtelen listájában a jó sztring vagy megelőzi b jó sztringet, vagy $a = b$, jelöljük $a \leq_R b$ formában.

1.2. Dinamikus programozás alkalmazása

Vezessük be a $\sigma^k = \underbrace{\sigma\sigma\cdots\sigma}_{k \text{ darab}}$ jelölést ($\sigma^0 = e$). Jelentse $L_{i,j}$ azon $2j$ hosszúságú jó szavak halmazát, melyeknek $(<)^i$ prefixe. Legyen $n_{i,j} = |L_{i,j}|$. Vegyük észre, hogy $L_{0,j}$ a pontosan $2j$ hosszúságú jó szavak halmaza. Az 1. lemma miatt

$$\bigcup_{j=1}^{\infty} L_{0,j} = L.$$

A $\leq_R$ rendezés definíciója szerint igazak az

$$\forall a \in L_{0,j};\ b \in L_{0,J};\ j \leq J: \qquad a \leq_R b, \tag{1}$$

$$\forall a \in L_{I,j};\ b \in L_{i,j} \setminus L_{I,j};\ i < I: \qquad a \leq_R b \tag{2}$$

állítások.

3. lemma. *Az $n_{i,j}$ a következő, rekurzív alakban adható meg:*

$$n_{i,j} = \begin{cases} n_{1,j} & \textit{ha } i = 0, & \text{(3a)} \\ 1 & \textit{ha } i = j = 1, & \text{(3b)} \\ 0 & \textit{ha } i > j, & \text{(3c)} \\ n_{i+1,j} + n_{i-1,j-1} & \textit{egyébként.} & \text{(3d)} \end{cases}$$

Bizonyítás. A 2. lemma miatt a (3a) eset triviális.

Ha $i = j = 1$, csak egy jó szót találhatunk, ez a $<>$. Tehát (3b) teljesül.

Ometeotl 1. feltétele miatt egy $2j$ hosszúságú jó szóban pontosan j dara $<$ jel van, több semmiképp sem lehet; ezzel (3c)-t is beláttuk.

Már csak $0 < i \leq j$ maradt, azaz a (3d) eset. Vegyük észre, hogy $i > 0$ esetén $L_{i,j}$ elemei kétfélék lehetnek: vagy $(<)^i<$, vagy $(<)^i>$ prefixük. Azon jó szavak, melyeknek $(<)^i< = (<)^{i+1}$ prefixe, pontosan az $L_{i+1,j}$ halmaz elemei. Tehát

$$\begin{aligned} n_{i,j} &= |L_{i,j}| \\ &= |L_{i+1,j}| + |L_{i,j} \setminus L_{i+1,j}| \\ &= n_{i+1,j} + |L_{i,j} \setminus L_{i+1,j}|, \end{aligned}$$

azaz elegendő

$$n_{i-1,j-1} = |L_{i,j} \setminus L_{i+1,j}|$$

bizonyítása. Ezt egy $L_{i-1,j-1} \longleftrightarrow L_{i,j} \setminus L_{i+1,j}$ bijekció megkonstruálásával látjuk be. Ugyanis ha egy $(<)^n>X$ alakú szó ($X \in \Sigma^*$) eleme $L_{i,j} \setminus L_{i+1,j}$-nek, akkor az $(<)^{i-1}X$ szó $2(j-1)$ hosszúságú és $L_{i-1,j-1}$ eleme; Ometeotl mindkét szabálya érvényes marad. Analóg, ellenkező irányú állítás is belátható. Következésképp a két halmaz elemeinek száma azonos. □

A 3. lemma szerint, dinamikus programozás felhasználásával könnyűszerrel kiszámíthatjuk $n_{i,j}$ értékeit. A Segédtömb-Kiszámol eljárás épp ezt teszi, egy $max + 1 \times max + 1$ méretű N tömb segítségével. Az eljárás végén az $N[i,j] = n_{i,j}$ lesz minden $0 \leq i \leq max$, $0 \leq j \leq i$ indexre.

```
Segédtömb-Kiszámol(N, max)
N[0,0] = 0
for i = 1 to max
    N[i,i] = 1
for j = 1 to max
    N[0,j] = N[1,j]
    for k = j + 1 to max
        N[k − j, k] = N[k − j + 1, k] + N[k − j − 1, k − 1]
```

A Segédtömb-Kiszámol a dinamikus programozásban megszokott „táblázat kitöltést” nem soronként, hanem átlósan végzi el:

- Először az 1. sor és a 2–3. sorok ciklusa kitölti a főátló elemeit, (3b) és (3c) figyelembe vételével.
- Ezután a 4–7. sorok ciklusa a táblázat bal alsó sarka felé haladva kitölti a többi átlót.
- A (3a) esetet az 1. és 5. sorok kezelik.

Erre a haladásra azért van szükség, mert (3d) szerint egy elem a fölötte balra illetve a tőle jobbra lévő elemtől is függ.

A futásidő az egymásba ágyazott a 4–7. és a 6–7. sorbeli ciklusok miatt $O(max^2)$.

1.3. A kódolás lépései

Mivel a kódolás egyéb lépései lényegében triviálisak, elegendő azzal foglalkozni, hogyan alakítható egy tetszőleges ν szám valamely x jó szóvá; azaz hogy találhatjuk meg az x jó szót, hogy pontosan ν darab olyan x' jó szó legyen, melyre $x' \leq_R x$. Így a feladatban szereplő választás az R listából mindig a $\nu - 1$. szám választását fogja jelenteni; az R lista ugyanis 0-tól indexelt, míg a $<>$ szó $\nu = 1$-hez fog tartozni.

Először meghatározzuk x hosszát. Az (1) miatt $|x| = 2j$, ha

$$\sum_{k=0}^{j-1} n_{0,k} < \nu \leq \sum_{k=0}^{j} n_{0,k}. \tag{4}$$

Ezt a műveletet végzi a Kódolás-Szóhossz eljárás, mely a μ változóban megadja, hogy x a $2j$ hosszúságú jó szavak között hányadik helyen áll az R listában.

Kódolás-Szóhossz(N, ν, μ)

```
j = 0
sum = 0
while sum + N[0, j] < ν
      sum = sum + N[0, j]
      j = j + 1
μ = ν − sum
return j
```

Azt, hogy a $2j$ hosszúságú jó szavak közül melyik a μ., egy rekurzió segítségével állapítjuk meg. Az alapötlet a következő: legyen $x = (<)^i{>}X$. Ekkor a (2) egyenlőtlenség és a 3. lemma bizonyításában vázolt bijekció miatt

$$\mu = n_{i+1,j} + \mu', \tag{5}$$

ahol μ' azt mutatja, hogy a $x' = (<)^{i-1}X$ szó hányadik helyen áll a $2j-2$ hosszúságú szavak között. Természetesen elvárjuk, hogy

$$0 < \mu' \leq n_{i-1,j-1}. \tag{6}$$

A $j = 1$ esetet némiképp speciálisan kell kezeljük, ott (5) nem alkalmazható, mivel az X az üres szó lenne, amely nem jó szó. Szerencsére $j = 1$ esetén csak $\mu = 1$ lehetséges, melyre $i = 1$, vagyis a keresett szó a $<>$.

4. lemma. *A j, μ változók, valamint az* (5) *és* (6) *egyenletek egyértelműen meghatározzák i értékét.*

Bizonyítás. Alkalmazzunk indirekt bizonyítást! Tegyük fel, hogy $i = i_1$ és $i = i_2$ kielégítik az (5) és (6) egyenleteket.

Az általánosság megszorítása nélkül feltehetjük azt is, hogy $i_2 = i_1 + 1$. (Ha $i_1 < i_2$, felcseréljük őket; ha pedig $i_2 - i_1 > 1$, akkor bármely $i_1 < i_2' < i_2$ is megfelel a feltételeknek.) Így azt kapjuk, hogy

$$0 < \mu - n_{i_1+1,j} \leq n_{i_1-1,j-1}, \tag{7}$$

$$0 < \mu - n_{i_2+1,j} = \mu - n_{i_1+2,j} \leq n_{i_2-1,j-1} = n_{i_1,j-1}, \tag{8}$$

azaz

$$n_{i_1+1,j} - n_{i_1+2,j} < \mu - n_{i_1+2,j} \leq n_{i_1,j-1}. \tag{9}$$

Ha $i_1 < j$, (3d) miatt

$$n_{i_1+1,j} < n_{i_1+2,j} + n_{i_1,j-1} = n_{i_1+1,j}, \tag{10}$$

ami ellentmondás.

Hátra van még az $i_1 = j$ eset, de akkor $i_2 > j$, azaz $(<)^{i_2}>$ nem lehet x prefixe. Vagyis ismét ellentmondásra jutottunk.

A fentiek szerint legfeljebb egy i (és hozzá tartozó μ') elégítheti ki a feltételeket. Mivel $x = (<)^i>X$ és $x' = (<)^{i-1}X$ jó szavak, ezért i és μ' is jól definiált, vagyis pontosan egy ilyen i létezik. □

A megfelelő i és μ' megkeresését végzi a Kódolás-Prefixhossz eljárás.

```
Kódolás-Prefixhossz(N, j, μ, μ')
if j = 1
    return 1
i = j
while N[i, j] < μ
    i = i − 1
if i = j
    μ' = μ − 0
else μ' = μ − N[i + 1, j]
return i
```

Ennek a két eljárásnak a segítségével könnyedén előállíthatjuk az x jó szó ν-höz. Először meghatározzuk ν-höz $j^{(1)}$ és $\mu^{(1)}$ értékét Kódolás-Szóhossz segítségével (a felső index az első iterációt jelöli). Majd Kódolás-Prefixhossz megadja $x =$

j	μ	i	μ'	x
3	4	1	1	<>????
2	1	2	1	<><<>?
1	1	1		<><<>>

1. táblázat. A KÓDOLÁS-PREFIXHOSSZ eljárás hívásai a $\nu = 7$ esetben.

$x^{(1)}$ egy prefixét közvetett módon $i^{(1)}$ és $\mu'^{(1)}$ változókkal. Az x szó egészét úgy kaphatjuk meg, hogy meghatározzuk $x' = x^{(2)}$-t; így következik az újabb iteráció $j^{(2)} = j^{(1)} - 1$ és $\mu^{(2)} = \mu'^{(1)}$ bemenetekkel. Pontosabban: összefűzve az $(<)^{i^{(1)}}>$ és az $x^{(2)}\left[i^{(1)}..2j^{(2)}\right]$ sztringeket kapjuk meg $x^{(1)}$-et. A rekurziót az indexek növelésével folytathatjuk a k. iterációig, amíg $j^{(k+1)} = 0$ nem lesz.

Ezt a módszert követi a KÓDOLÁS eljárás.

KÓDOLÁS(N, ν)
1 $j^{(1)} =$ KÓDOLÁS-SZÓHOSSZ$(N, \nu, \mu^{(1)})$
2 $i^{(0)} = 1$
3 $k = 1$
4 **while** $j^{(k)} > 0$
5 $\quad i^{(k)} =$ KÓDOLÁS-PREFIXHOSSZ$(N, j^{(k)}, \mu^{(k)}, \mu^{(k+1)})$
6 $\quad$ **for** $\ell = 1$ **to** $i^{(k)} + 1 - i^{(k-1)}$
7 $\quad\quad$ **print** „<”
8 $\quad$ **print** „>”
9 $\quad j^{(k+1)} = j^{(k)} - 1$
10 $\quad k = k + 1$

A ciklus a 6-8. sorokban mindig kiírja x már megismert prefixét, a 2. sor pedig az első iteráció speciális kezelését kerüli el.

Példának vegyük a $\nu = 7$ esetet. Először a KÓDOLÁS-SZÓHOSSZ futtatásával kapjuk, hogy $j^{(1)} = 3$ és $\mu^{(1)} = 4$. Az 1. táblázat mutatja a KÓDOLÁS-PREFIXHOSSZ hívások eredményeit.

1.4. A dekódolás lépései

A dekódolás a kódolás lépéseinek fordított sorrendű elvégzésével történik. Pontosabban: a KÓDOLÁS-PREFIXHOSSZ eljárás 7. és 8. sorában, valamint a KÓDOLÁS-SZÓHOSSZ eljárás 6. sorában levont értékek kiszámítása és összeadása után kapható meg ν értéke egy adott x-hez.

Ezt a műveletet a DEKÓDOLÁS eljárás végzi.

DEKÓDOLÁS(N, x)

1 $\nu = 1$
2 $k = 1$
3 $j^{(1)} = \frac{|x|}{2}$
4 $i^{(1)} = 0$
5 **for** $p = 1$ **to** $|x|$
6 **if** $x[p] ==$ „<”
7 $i^{(k)} = i^{(k)+1}$
8 **elseif** $x[p] ==$ „>”
9 **if** $i^{(k)} < j^{(k)}$
10 $\nu = \nu + N\left[i^{(k)} + 1, j^{(k)}\right]$
11 **else** $\nu = \nu + 0$
12 $\nu = \nu - N\left[0, j^{(k)} - 1\right]$
13 $i^{(k+1)}) = i^{(k)} - 1$
14 $j^{(k+1)}) = j^{(k)} - 1$
15 $k = k + 1$
16 **return** ν

A változók felső indexes jelölése csupán az érthetőség kedvéért történik, az indexek megfelelnek a KÓDOLÁS eljárás során megjelenőknek. A $\nu = 1$ kezdőérték az 1. sorban azt biztosítja, hogy magát a dekódolt szót is beleszámítsuk ν értékébe annak definíciója szerint (hiszen $x \leq_R x$). A 10. és 11. sorok a KÓDOLÁS-PREFIXHOSSZ, míg a 12. sor a KÓDOLÁS-SZÓHOSSZ eljárás által levont értékeket adja hozzá ν-höz.

2. Az implementáció

Az algoritmusokat C++ nyelven implementáltam, a fordításhoz GCC (`g++`) fordítóprogramot használtam.

A programkódom (`main.cxx`) a KÓDOLÁS és DEKÓDOLÁS eljárásokon kívül természetesen tartalmazza a bementet beolvasásához, illetve a betűcsoportok, számok és L-beli szavak egymásba alakításához szükséges eljárásokat is.

A program alapesetben a bemenetet a `S65.be` fájlból olvassa, a kimenetet pedig a `S65.ki` fáljba írja. Ez a fordításkor bizonyos preprocesszor-szimbólum definiálásával írható felül, melyeket a 2. táblázat tartalmaz.

kapcsoló	leírás	alapérték
`-DBE=''`*fájlnév*`''`	Bemeneti fájl neve.	`S65.be`
`-DKI=''`*fájlnév*`''`	Kimeneti fájl neve.	`S65.ki`
`-DSTDIO=1`	Standard ki- és bemenet használata a fájlok helyett.	kikapcsolt

2. táblázat. Kapcsolók az I/O műveletek céljainak módosításához.