S. 65.

Adrián Patrik, 12. évf.
Baross Gábor Középiskola, Szakiskola és Kollégium, Debrecen

A feladat két részfeladatból áll: az egyik egy számrendszer-konverzió, a másik a szabályos zárójelezések lexikografikus rendezéséről szól. Kezdjük a az előbbivel!

Egy m alapú számrendszerben bármely x szám értelmezése a következő:

$$x = \sum_{i=-\infty}^{\infty} m^i d_i$$

ahol d_i az i-edik helyiértéken álló számjegy. Nincs ez másként az esetünkben alkalmazott 27-es számrendszerben sem; a megszokástól annyiban térünk el, hogy számjegyek helyett már a nulla értéktől betűket ill. a 26 helyett a szóközt használjuk. Az egész számokban a helyiértékek jobbról balra nőnek, ami éppen fordított a sztringek indexelésével. Ezért ha a szám b hosszú, akkor

$$x = \sum_{i=0}^{b-1} s_i \cdot 27^{b-i-1}$$

Visszafelé konvertálni az ismert algoritmussal lehet, ahol a szám utolsó számjegye az alappal képzett osztási maradéka, a további részét pedig rekurzívan állítjuk elő a szám alappal vett hányadosának egészrészéből. Ez az egyes blokkokat fordított sorrendben adja vissza, tehát az eredményül kapott sztringeket a kiírás előtt meg kell fordítani.

Egy 12 jegyű, 27-es számrendszerbeli szám akkor veszi fel a maximális értékét, ha mind a 12 jegye maximális, azaz 26. Ekkor az értéke

$$\sum_{i=0}^{11} 26 \cdot 27^i = 27^{12} - 1 = 150\ 094\ 635\ 296\ 999\ 120$$

Ez még elfér egy 64 bites egészben.

A kódsorozatok és az egész számok halmazai közötti konverzió ennél nehezebb kérdés. Azt azért nem nehéz észrevenni, hogy a jelsorozatokra érvényes két szabály a szabályosan zárójelezéseket határozza meg. Az egyenlőségi feltétel miatt mindkét jelből ugyanannyinak kell lennie (minden nyitó zárójelhez tartozzon záró pár és fordítva), a megelőzési szabály pedig azt garantálja, hogy egy pár nyitó tagja előbb szerepel a zárónál. Mivel az egyes párokat minőségileg nem különböztetjük meg, ezért a mennyiségi kritérium elegendő.

Tegyük fel, hogy a k-adik szabályos zárójelezést keressük, ahol a sorrend a megadott reláció alapján értendő. Mint általában, itt is megköveteljük, hogy k-t nullától értelmezzük, vagyis a hagyományos értelemben vett első elem itt a nulladik lesz. A megszokás mellett még azzal is érvelhetünk a döntés mellett, hogy a szöveges- és a számalak konverziójában is a legkisebb reprezentálható érték a 0.

Az összes lehetőség előállítása, ezek rendezése, majd ebből a k-adik meghatározása reménytelennek ígérkezik, mert az n hosszú szabályos zárójelsorozatok (továbbiakban *szavak*) száma n exponenciális függvénye. Az algoritmusnak tehát céltudatosan a k-adik szót kell keresnie. Célszerű volna egy olyan algoritmust keresni, ami képes arra, hogy a szót az elejétől a végéig lépésekben előállítsa. Alkalmas célfüggvénnyel a dinamikus programozás módszere alkalmazható.

A szó első karaktere mindig <. Ez után mi állhat? Minden esetben egy olyan szószelet, amely majdnem szabályos, de valahol szerepel benne ennek a nyitó jelnek a záró párja. Ha ennek a szószeletnek a kezdőjele ismét <, akkor ezután olyan szelet következik, amiben két extra záró szimbólum van. Ha ugyanakkor az elején > áll, akkor visszaállt az egyensúly, újból egy szabályos szó következik. Azok a szavak természetesen, amelyekben a második helyen < szerepel, előbb állnak, mint azok, amelyekben >. Ezt a gondolatot kellene valahogyan formalizálni.

S. 65.

Adrián Patrik, 12. évf.
Baross Gábor Középiskola, Szakiskola és Kollégium, Debrecen

Legyen a d függvény értelmezése a következő: Jelölje $d(n, m)$ az n hosszú zárójelsorozatok közül azokat, amelyekben m-mel több záró jel szerepel, mint nyitó. Általánosan $d(n, m) = d(n-1, m+1)+d(n-1, m-1)$, ami esetszétválasztással igazolható. Ha az n hosszú szó(szelet) első szimbóluma `<`, akkor utána egy olyan $n-1$ hosszú szószelet áll, amiben eggyel több záró jel van, mert az ennek a nyitónak is benne kell lennie a párjának. Ugyanakkor ha az első szimbólum `>`, akkor egy záró jelet már elintéztünk, tehát a következő $n-1$ hosszú részbe már csak eggyel kevesebb kell.

A következő határesetekkel kell számolni:

- $d(n, m) = 0$ ha $m > n$ vagy $m < 0$, hiszen nem állhat több záró jel egy szószeletben, mint amennyi annak hossza, és a negatív darabszám is nehezen értelmezhető.
- $d(0, 0) = 1$, mert az egyetlen nulla hosszú szóban (üres sztring) pontosan ugyanannyi nyitó szimbólum van mint záró.

Ezekből a szabályokból már elő is állítható a d függvény értéke. Hogy a negatív paraméterekkel ne kelljen törődni, a $d(n, 0) = d(n-1, -1) + d(n-1, 1) = d(n-1, 1)$ esetet külön kezeljük. Megjegyezzük, hogy a $d(2n, 0)$ értéke az n-edik katalán szám, C_n.

Térjünk vissza a k-adik szó előállítására, amit ezentúl w_k-val jelölünk. Először célszerű meghatározni, hogy w_k milyen hosszú. Ezt úgy tudjuk megtenni, hogy k értékét rendre csökkentjük $C_1, C_2, \ldots$ értékével, egészen addig, míg az eredmény nemnegatív. A soron következő, ki nem vont katalán szám indexe a w_k-ban szereplő zárójelpárok számát adja, aminek $|w_k|$ a kétszerese.

Az előállítást egy olyan $G(n, m, k)$ algoritmussal végezzük, amelynek paraméterei az előállítandó szószelet hossza, a záró szimbólumok többletszáma és a szükséges pozíció. Nyitó jelet akkor írhatunk le, ha az ehhez tartozó szuffixumok száma legalább k, vagyis $d(n-1, m+1, k) >= k$, más szóval legalább az első k megfelelő paraméterekkel rendelkező szószelet `<` szimbólummal kezdődik. Ha nem, akkor `>` kell, hogy a szószelet elejére kerüljön. Mivel azonban így átugrottuk a `<` jellel kezdődő szószeleteket, ezért ezek szuffixumai között már nem a k-adikat, hanem a $k - d(n-1, m+1)$-ediket kell keresni. G a $(0, 0, 0)$ számhármasra ér véget.

A dekódolás ugyanezen elven alapszik: Végigkövetjük, hogy a kódoló algoritmus milyen szimbólumokat írt ki, azaz ha `>` jelet, akkor összegezzük a lexikografikusan előbb álló, tehát `<` jellel kezdődő szuffixumok számát; ez az összeg az összes pozíción véve, majd hozzáadva a rövidebb szavak számát is (mert a rendezés szerint azok is előbb állnak, mint a megadott szó) megkapjuk a keresett k értékét. (Az algoritmusnak szükséges összes többi paramétert ismerjük.) Ebből a visszakódolt értékből a már korábban ismertetett módszerrel visszaállíthatjuk a szöveges alakot.

A C# nyelven készült program a Microsoft Visual C# 2010-es verziójának fordítóprogramjával, a `csc s65.cs` paranccsal fordítható. (A forráskód UTF-8 kódolású.)