# Project Euler 219 - Feladat megoldás

Poór Boldizsár

## 1 Feladatleírás

Let A and B be bit strings (sequences of 0's and 1's).
If A is equal to the leftmost length(A) bits of B, then A is said to be a prefix of B.
For example, 00110 is a prefix of 001101001, but not of 00111 or 100110.

A prefix-free code of size n is a collection of n distinct bit strings such that no string is a prefix of any other. For example, this is a prefix-free code of size 6:

0000, 0001, 001, 01, 10, 11

Now suppose that it costs one penny to transmit a '0' bit, but four pence to transmit a '1'.
Then the total cost of the prefix-free code shown above is 35 pence, which happens to be the cheapest possible for the skewed pricing scheme in question.
In short, we write Cost(6) = 35.

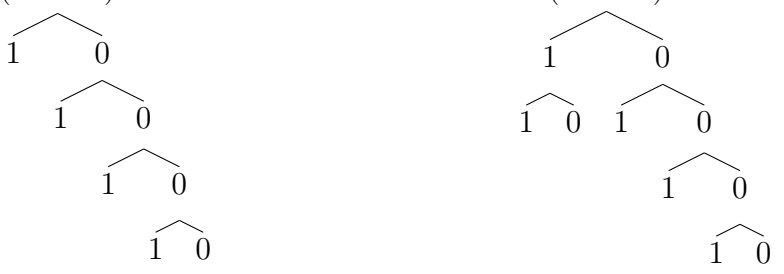
What is Cost(109) ?

## 2 Megoldás

### 2.1 $\mathcal{O}(2^n)$-es megoldás:

Az egyszerű megközelítés az, hogy csinálunk egy fát, oly módon, hogy mindig megkeressük, melyik a legkisebb értékű levél és ahhoz csatolunk két levelet. Az egyik lesz az 1, a másik a 0.

FROM: (ROOT) → TO: (ROOT)




A fát felépíteni $\mathcal{O}(2^n)$ költséggel lehet, ami $10^9$-es n-el $2^{90} = C \cdot 1237940039285380274899124224$, ami elég sokáig tartana.

## 2.2 $\mathcal{O}(n \cdot \log n)$-es megoldás:

Egy hatékonyabb megoldáshoz az kell, hogy az előző, pazarló megoldásból a felesleges információk feldolgozását és eltárolását hagyjuk ki.
Vegyük észre, hogy értelmetlen eltárolni a teljes fát, valamint azt is, hogy a feladat megoldása szempontjából teljesen értelmetlen eltárolni a kódokat, elég csak az árukat. Az adat tároláshoz egy prioritási-sor (priority-queue) tökéletesen megfelel, sőt, így még a legkisebb elem megtalálása is meg van oldva $\mathcal{O}(\log n)$ alatt.
A programban, minden alkalommal, amikor kiveszünk a sorból egy elemet, visszateszünk két másikat, az egyik a kivett elem 1-el a másik 4-el megnövelt értéke.
A program így belátható időn belül le fog futni, azonban még mindig lehet mit javítani.

## 2.3 $\mathcal{O}(\log n)$ körüli megoldás:

Annak érdekében, hogy hatékonyabbak algoritmust írjunk, észre kell vennünk, hogy még mindig nem használunk ki minden információforrást:
Gondoljunk bele, hogy a sorban most nagyon sok azonos elem van. Ha számon tartanánk, mennyi azonos elem van a sorban, és egyszerre végeznénk el az összes kivételt majd visszarakást, elképesztően sok számítást és memóriát megspórolhatnánk magunknak.
A gyors megoldás lényegi pontja ez, de hogy jobb programot csináljunk, azt is vegyük észre, hogy az adattárolónknak összesen 4 hosszúnak kell lennie, ha $n > 3$. Hogy elkerüljük a felesleges elágazást a ciklusunkon belül, kivisszük abból, és ha $n > 3$, egyszerűen visszatérhetünk az előre - fejben - kiszámolt értékkel.
A megértés megkönnyítéséért megvizsgálhatjuk a táblázatot, ami a négy változónk értékeit mutatja, a ciklusszámunk függvényében.

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1. | 1 | | | 1 | | | | | | | | | |
| 2. | | 1 | | 1 | 1 | | | | | | | | |
| 3. | | | 1 | 1 | 1 | 1 | | | | | | | |
| 4. | | | | 2 | 1 | 1 | 1 | | | | | | |
| 5. | | | | | 3 | 1 | 1 | 2 | | | | | |
| 6. | | | | | | 4 | 1 | 2 | 3 | | | | |
| 7. | | | | | | | 5 | 2 | 3 | 4 | | | |
| 8. | | | | | | | | 7 | 3 | 4 | 5 | | |
| 9. | | | | | | | | | 10 | 4 | 5 | 7 | |
| 10. | | | | | | | | | | 14 | 5 | 4 | 10 |

Ami még megfontolandó kérdés, hogy hogyan tudjuk, mikor eljutottunk $n$-hez, hisz itt most nagyon nagyokat ugrálunk minden ciklusban.
Hogy ez ne jelentsen problémát, a ciklusban meg kell vizsgálni, hogy $i$, vagy a legkisebb értékű elem darabszáma a kisebb-e? Amikor a darabszám kisebb, akkor miután megnöveltük a legkisebbnél 1-el és 4-el nagyobb elemek darabszámát a legkisebb darabszámával, töröljük azt. Amikor $i$ a kisebb, elvégezzük a szokásos eljárást a törléstől eltekintve. A helyett csökkentjük a darabszámot $i$-vel. A ciklus lefutása után már csak összegeznünk kell az eredményeket.

Listing 1: Project Euler 219 - Effektív megoldás Python-ban

```
def func(i):
    if i > 3:
        # A tablazatban a 3. pont allapota
        dictionary = {3: 1, 4: 1, 5: 1, 6: 1}
        i -= 4
        while i:
            smallest = min(dictionary.keys())
            if i > dictionary[smallest]:
                quantity = dictionary.pop(smallest)
                dictionary[smallest + 1] += quantity
                dictionary[smallest + 4] = quantity
                i -= quantity
            else:
                dictionary[smallest] -= i
                dictionary[smallest + 1] += i
                dictionary[smallest + 4] = i
                i = 0
        summary = 0
        for price, count in dictionary.items():
            summary += price * count
        return summary
    else:
        return [0, 1, 5, 11][i]


print(func(10**9))
```