Tento problém se dá řešit několika způsoby. Řešení, které jsme zvolili my a které si zde popíšeme, je nejrychlejší možné, ale zároveň je snadné na pochopení i implementaci. Bohužel za tuto rychlost platíme zvýšenými nároky na paměť. Myšlenka řešení je předem si předpočítat výsledky pro všechny možné vstupy.

Než si popíšeme řešení, ve stručnosti si zopakujeme zadání problému. Mějme číslo n a následující operace:

1. Zvětši n o 1.
2. Zmenši n o 1.
3. Pokud je n dělitelné dvěmi, vyděl ho.
4. Pokud je n dělitelné třemi, vyděl ho.

Cílem je dostat číslo 1 na co nejmenší počet operací. Pokud bychom neuvažovali krok č. 1, předpočítat si řešení je triviální. Naštěstí, jak si později dokážeme, se nám nikdy nevyplatí zvýšit n dvakrát po sobě. Řešení se dá tedy popsat funkcí jako:

$$f(n) = min \begin{cases} f(n-1)+1 & \\ f(n/2)+1 & \text{když } n \text{ je dělitelné dvěmi} \\ f(n/3)+1 & \text{když } n \text{ je dělitelné třemi} \\ f((n+1)/2)+2 & \text{když } n+1 \text{ je dělitelné dvěmi} \\ f((n+1)/3)+2 & \text{když } n+1 \text{ je dělitelné třemi} \end{cases}$$

S počáteční podmínkou $f(1) = 0$. Všimněte si, že v popisu funkce jsou všechny argumenty na pravé straně menší než n.

Teď si uvedeme důkaz, proč se nám nevyplatí zvýšit n dvakrát (nebo vícekrát) po sobě. Nejdřívě uvažujme pouze operaci č. 3. Pokud je číslo n dělitelné dvěmi, tak je jistě dělitelné i číslo $n+2$. Pokud obě čísla vydělíme, dostaneme čísla n/2 (na jednu operaci) a $\frac{n+2}{2}$ (na 3 operace). Nicméně číslo $\frac{n+2}{2}$ jsme schopni dostat na operace 2 – nejdříve vydělit n dvěmi a poté přičíst 1. Tudíž v tomto případě se nám nemůže vyplatit dvakrát inkrementovat.

Pokud n není dělitelné dvěmi, musíme přičíst alespoň 3, nicméně číslo $\frac{n+3}{2}$ můžeme dostat na 3 kroky (přičíst 1, vydělit 2 a opět přičíst 1), tudíž toto se nám opět nevyplatí.

Nyní uvažujme operaci č. 4. Pokud n je dělitelné třemi, tak dostáváme stejný případ jako v předcházejícím kroku. Opět je výhodnější nejdříve n vydělit a poté přičíst 1, než přičíst třikrát 1 a až poté vydělit.

Pokud číslo n dává zbytek 1 po dělení třemi, tak číslo $\frac{n+2}{3}$ dostaneme na tři operace, ať nejdříve odečteme jedničku, poté ho vydělíme 3 a opět přičteme 1 nebo nejdříve přičteme dvakrát 1 a až poté ho vydělíme třemi. Opět vidíme, že zvýšením čísla dvakrát nezískáme žádnou výhodu.

Nakonec pokud číslo n dává zbytek 2 po dělení třemi, tak bychom potřebovali alespoň čtyřikrát inkrementovat n a až poté ho vydělit, což se nám zřejmě opět nevyplatí.

Ve výše zmíněných případech jsme si ukázali, že se nám nikdy nevyplatí několikrát po sobě zvýšit číslo n.

Celé řešení problému, se dá tedy vyjádřit v následujících pár řádcích (funkce $f(n)$ byla zmíněna výše).

```
byte reseni[10^8 +1]
reseni[0] = 0
for ( i = 1; i<=10^8; i++ )
  reseni[i] = f(i)
```

Na závěr jenom zmíňme, že i když paměťové nároky jsou poměrně vysoké (potřebujeme si pamatovat 10^8 čísel), velikost každého z čísel nepřekročí $2 * log_2(10^8) = 53$.