Druge vaje APS2: deli in vladaj

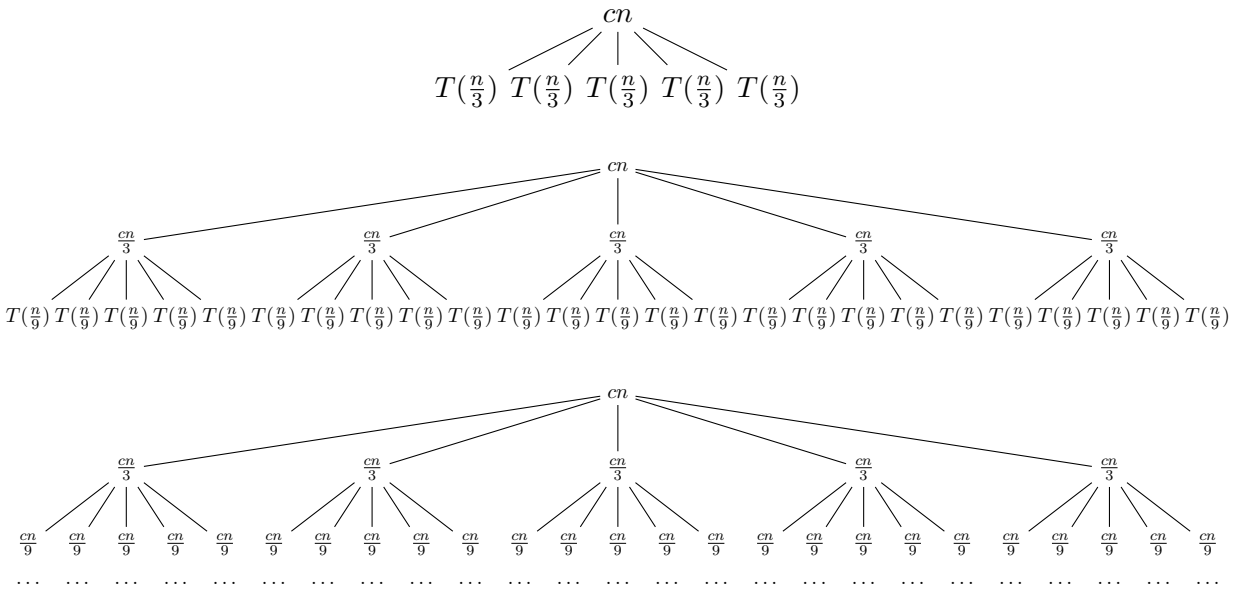
Naloge

(1) S pomočjo drevesne metode uganite rešitev enačbe $T(n) = 5T(n/3) + \Theta(n)$ in jo dokažite s pomočjo substitucijske metode. □

Enačbo $T(n) = 5T(n/3) + \Theta(n)$ zapišimo kot $T(n) = 5T(n/3) + cn = cn + 5T(n/3)$ in jo vstavljajmo samo vase:

$$\begin{aligned} T(n) &= cn + 5T(n/3) \\ &= cn + 5(cn/3 + 5T(n/9)) = cn + 5cn/3 + 25T(n/9) \\ &= cn + 5cn/3 + 25(cn/9 + 5T(n/27)) = cn + 5cn/3 + 25cn/9 + 125T(n/27) \\ &= \dots \end{aligned}$$

Postopek lahko prikažemo z drevesom (od tod ime *drevesna metoda*):



Rekurenca se izteče, ko argument funkcije $T(.)$ pade pod neko konstantno vrednost. Zaradi enostavnosti bomo vzeli kar 1, a tudi če bi rekurenco »odrezali« pri neki konstanti r, bi na koncu dobili isti rezultat. No, če kot ciljni argument funkcije $T(.)$ vzamemo 1, se rekurenca izteče, ko je $n/3^k \leqslant 1$, zato je število nivojev drevesa enako $k = \log_3 n$. Na nivoju listov imamo konstantno porabo časa (npr. d za vsak list), število listov pa je enako $5^{\log_3 n} = (n^{\log_n 5})^{\log_3 n} = n^{\log_3 n \log_n 5} = n^{\log_3 5}$.

Skupna poraba časa je torej

$$\begin{aligned} T(n) &= \underbrace{cn + 5cn/3 + 25cn/9 + 125cn/27 + \dots}_{\log_3 n \text{ členov}} + dn^{\log_3 5} \\ &= cn \left(\underbrace{1 + \frac{5}{3} + \left(\frac{5}{3}\right)^2 + \left(\frac{5}{3}\right)^3 + \dots}_{\log_3 n \text{ členov}} \right) + dn^{\log_3 5} \\ &= cn \sum_{i=0}^{\log_3 n - 1} \left(\frac{5}{3}\right)^i + dn^{\log_3 5}. \end{aligned}$$

Dobili smo divergentno geometrijsko vrsto, zato je ne moremo »podaljšati« v neskončnost in si tako poenostaviti računanje. Lahko pa si pomagamo s formulo

$$\sum_{i=0}^{k} p^i = \frac{p^{k+1}-1}{p-1}$$

in mrcvarimo naprej:

$$\begin{aligned}
T(n) &= cn \sum_{i=0}^{\log_3 n - 1} \left(\frac{5}{3}\right)^i + dn^{\log_3 5} \\
&= cn \frac{(\frac{5}{3})^{\log_3 n} - 1}{\frac{5}{3} - 1} + dn^{\log_3 5} \\
&= Kcn \left(\left(\frac{5}{3}\right)^{\log_3 n} - 1 \right) + dn^{\log_3 5} \\
&= Kcn \frac{5^{\log_3 n}}{3^{\log_3 n}} - Kcn + dn^{\log_3 5} \\
&= Kcn \frac{5^{\log_3 n}}{n} - Kcn + dn^{\log_3 5} \\
&= Kc \cdot 5^{\log_3 n} + dn^{\log_3 5} - Kcn \\
&= Kcn^{\log_3 5} + dn^{\log_3 5} - Kcn \\
&= (Kc + d)n^{\log_3 5} - Kcn \\
&= \Theta(n^{\log_3 5}).
\end{aligned}$$

Dobili smo kandidatno rešitev $T(n) = \Theta(n^{\log_3 5})$, vendar pa jo moramo še preveriti s substitucijsko metodo. Najprej dokažimo $T(n) = O(n^{\log_3 5})$. Postavimo hipotezo

$$T(n) \leqslant cn^{\log_3 5} \text{ za vsak } n \geqslant n_0$$

in jo dokažimo z indukcijo. Predpostavimo, da hipoteza velja za vsak $n' < n$ (torej tudi za $n' = n/3$), in to predpostavko vstavimo v originalno rekurenčno enačbo:

$$\begin{aligned}
T(n) &= 5T\left(\frac{n}{3}\right) + \Theta(n) \\
&\leqslant 5c\left(\frac{n}{3}\right)^{\log_3 5} + \Theta(n) \\
&= 5c\frac{n^{\log_3 5}}{3^{\log_3 5}} + \Theta(n) \\
&= 5c\frac{n^{\log_3 5}}{5} + \Theta(n) \\
&= cn^{\log_3 5} + \Theta(n).
\end{aligned}$$

Tristo kosmatih, saprabolt nazaj! Tako smo se mučili, a nam hipoteze $T(n) \leqslant cn^{\log_3 5}$ ni uspelo dokazati. Ovbe, ovbe ...

No, ni tako hudo. K sreči si lahko pomagamo s standardnim trikom in postavimo močnejšo hipotezo:

$$T(n) \leqslant cn^{\log_3 5} - dn \text{ za vsak } n \geqslant n_0.$$

Sedaj nam uspe:

$$\begin{aligned}
T(n) &= 5T\left(\frac{n}{3}\right) + \Theta(n) \\
&\leqslant 5\left(c\left(\frac{n}{3}\right)^{\log_3 5} - \frac{dn}{3}\right) + \Theta(n) \\
&= 5c\frac{n^{\log_3 5}}{3^{\log_3 5}} - \frac{5dn}{3} + \Theta(n) \\
&= 5c\frac{n^{\log_3 5}}{5} - \frac{5dn}{3} + \Theta(n) \\
&= (cn^{\log_3 5} - dn) + \left(\Theta(n) - \frac{2dn}{3}\right) \\
&\leqslant cn^{\log_3 5} - dn \text{ za vsak } n \geqslant n_0,
\end{aligned}$$

saj lahko konstanto d nastavimo tako, da bo izraz $(2dn/3)$ za vsak $n \geqslant n_0$ večji od linearne funkcije $kn + \ell$, ki se skriva v $\Theta(n)$.

Pri dokazovanju trditve $T(n) = \Omega(n^{\log_3 5})$ uporabimo hipotezo $T(n) \geqslant cn^{\log_3 5} + dn$ (ali kar $T(n) \geqslant cn^{\log_3 5}$) in jo dokažemo na analogen način.

(2) Kolikšna je časovna zahtevnost sledečega algoritma v odvisnosti od dolžine tabele A?[1]

```
function Stooge-Sort(A)
    Stooge-Sort(A, 0, |A| − 1)

function Stooge-Sort(A, p, r)
    if A[p] > A[r] then
        med seboj zamenjaj A[p] in A[r]
    if p + 1 < r then
        k ← ⌊(r − p + 1) / 3⌋
        Stooge-Sort(A, p, r − k)
        Stooge-Sort(A, p + k, r)
        Stooge-Sort(A, p, r − k)
```

□

Funkcijo najprej rekurzivno pokličemo na prvih dveh tretjinah tabele, nato na drugih dveh tretjinah, nazadnje pa še enkrat na prvih dveh tretjinah tabele. Funkcija se torej trikrat izvede na dveh tretjinah tabele. Ker za razdelitev problema na podprobleme in združevanje rešitev podproblemov v rešitev izhodiščnega problema porabimo zgolj konstantno mnogo časa, lahko časovno zahtevnost v odvisnosti od $n = |A|$ zapišemo kot

$$T(n) = 3T\left(\frac{2n}{3}\right) + \Theta(1).$$

Po krovnem izreku je $a = 3$, $b = \frac{3}{2}$ in $d = 0$. Ker je $a > b^d$, je časovna zahtevnost enaka

$$T(n) = \Theta(n^{\log_b a}) = \Theta(n^{\log_{3/2} 3})$$

oziroma $T(n) = o(n^{2,71})$ in $T(n) = \omega(n^{2,70})$.

(3) Dokažite, da časovna zahtevnost sledečega algoritma narašča kvečjemu tako hitro kot funkcija 2^n in vsaj tako hitro kot funkcija n^2.[2]

[1] T. Cormen *et al.*, str. 202.

[2] A. Broder & J. Stolfi. Pessimal Algorithms and Simplexity Analysis. *ACM SIGACT News*, 16(3): 49–53, 1984.

```
function SLOWSORT(A)
    SLOWSORT(A, 0, |A| − 1)

function SLOWSORT(A, p, q)
    if p < q then
        m ← ⌊(p + q) / 2⌋
        SLOWSORT(A, p, m)
        SLOWSORT(A, m + 1, q)
        if A[q] < A[m] then
            med seboj zamenjaj A[m] in A[q]
        SLOWSORT(A, p, q − 1)
```

□

Časovno zahtevnost algoritma lahko zapišemo kot

$$T(n) = 2T(n/2) + T(n-1) + \Theta(1).$$

Najprej pokažimo, da velja $T(n) = O(2^n)$. Postavimo hipotezo $T(n) \leqslant 2^n c$ za vsak $n \geqslant n_0$ in jo dokažimo z indukcijo:

$$\begin{aligned} T(n) &= 2T(n/2) + T(n-1) + \Theta(1) \\ &\leqslant 2(2^{n/2}c) + 2^{n-1}c + \Theta(1) \\ &= 2^{1+n/2}c + 2^{n-1}c + \Theta(1). \end{aligned}$$

Za dovolj velike n gotovo velja $2^{1+n/2}c \leqslant 2^{n-1}c - d$:

$$\begin{aligned} T(n) &\leqslant 2^{n-1}c + 2^{n-1}c - d + \Theta(1) \\ &= 2^n c - d + \Theta(1) \\ &\leqslant 2^n c. \end{aligned}$$

Pokažimo še, da velja $T(n) = \Omega(n^2)$. Postavimo hipotezo $T(n) \geqslant cn^2$ za vsak $n \geqslant n_0$ in jo dokažimo z indukcijo:

$$\begin{aligned} T(n) &= 2T(n/2) + T(n-1) + \Theta(1) \\ &\geqslant 2c(n/2)^2 + c(n-1)^2 + \Theta(1) \\ &= \frac{1}{2}cn^2 + cn^2 - 2cn + c + \Theta(1) \\ &= \frac{3}{2}cn^2 - 2cn + c + \Theta(1) \\ &\geqslant cn^2 \end{aligned}$$

Karacubovo množenje velikih števil

»Velika« števila so tista, ki so prevelika za standardni 32- ali 64-bitni celoštevilski podatkovni tip. Takšna števila običajno predstavimo kot zaporedje števk v desetiškem, dvojiškem ali katerem tretjem številskem sistemu. Ni težko videti, da lahko dve števili z n števkami zmnožimo v času $O(n^2)$, teoretična spodnja meja pa je seveda $\Omega(n)$, saj moramo vsako od $2n$ števk zmnožka vsaj zapisati. No, do $\Theta(n)$ ne bomo prišli, izkaže pa se, da lahko s preprostim trikom, ki ga je leta 1960 odkril Anatolij Aleksejevič Karacuba, pridelamo $o(n^2)$.

Predpostavili bomo, da sta števili, ki ju množimo, zapisani v desetiški obliki (kot zaporedji števk od 0 do 9), vendar pa bi algoritem zlahka prilagodili tudi za druge številske sisteme.

Če sta a in b enomestni števili, lahko njun zmnožek izračunamo v času $\Theta(1)$. V nasprotnem primeru pa predpostavimo, da sta obe števili n-mestni, kjer je n sodo število; če to ni res, ju pač dopolnimo z vodilnimi ničlami.[3] Števili zapišimo takole:

$$a = 10^{n/2}a_1 + a_0,$$
$$b = 10^{n/2}b_1 + b_0.$$

Je tak zapis že dovolj, da običajno časovno zahtevnost množenja ($\Theta(n^2)$) zmanjšamo? Poglejmo:

$$ab = 10^n a_1 b_1 + 10^{n/2}(a_1 b_0 + a_0 b_1) + a_0 b_0$$

Problem množenja n-mestnih števil torej razstavimo na štiri ($a = 4$) probleme velikosti $n/2$ ($b = 2$), za združevanje rezultatov podproblemov pa potrebujemo čas $\Theta(n)$ ($d = 1$); množenje s potenco števila 10 seveda ni »pravo« množenje, saj gre zgolj za zamikanje števk. Ker je $a > b^d$, je po krovnem izreku $T(n) = \Theta(n^{\log_b a}) = \Theta(n^{\log_2 4}) = \Theta(n^2)$. Šment!

No, k sreči se izkaže, da lahko s preprosto matematično telovadbo eno množenje prihranimo. Uvedimo

$$c_2 = a_1 b_1,$$
$$c_0 = a_0 b_0,$$
$$c_1 = (a_1 + a_0)(b_1 + b_0) - c_2 - c_0.$$

Ni težko pokazati, da velja

$$ab = 10^n c_2 + 10^{n/2} c_1 + c_0.$$

Ker za izračun koeficientov c_0, c_1 in c_2 potrebujemo zgolj tri množenja, smo problem množenja n-mestnih števil to pot razstavili na tri ($a = 3$) probleme velikosti $n/2$ ($b = 2$), za združevanje rezultatov podproblemov pa še vedno potrebujemo $\Theta(n)$ časa ($d = 1$). Vnovič je $a > b^d$, zato po krovnem izreku dobimo $T(n) = \Theta(n^{\log_b a}) = \Theta(n^{\log_2 3}) = O(n^{1,59})$. Jupiii!

Za lažje razumevanje algoritma zmnožimo števili $a = 123$ in $b = 45$ ($n = 4$). Velja $a_1 = 1$, $a_0 = 23$, $b_1 = 0$ in $b_0 = 45$. Izračunamo $c_2 = a_1 b_1 = 1 \cdot 0 = 0$, $c_0 = a_0 b_0 = 23 \cdot 45 = 1035$ ($c_2' = 2 \cdot 4 = 8$, $c_0' = 3 \cdot 5 = 15$, $c_1' = 5 \cdot 9 - 8 - 15 = 22$, $a_0 b_0 = 100c_2' + 10c_1' + c_0' = 800 + 220 + 15 = 1035$) in $c_1 = (a_1 + a_0)(b_1 + b_0) - c_2 - c_0 = 24 \cdot 45 - 0 - 1035 = 45$ (seveda tudi $24 \cdot 45$ izračunamo rekurzivno). Iskani zmnožek je torej $ab = 10^4 c_2 + 10^2 c_1 + c_0 = 10^4 \cdot 0 + 100 \cdot 45 + 1035 = 5535$.

Kako pa se Karacuba obnese v praksi? V skladu s pričakovanji, a le pod pogojem, da rekurzijo na primerni točki odrežemo in zmnožke »dovolj majhnih« (največ M-mestnih) števil izračunamo po klasičnem kvadratnem postopku. Pri moji implementaciji se je splačalo vzeti $M = 64$. S povečevanjem M se čedalje bolj čutijo negativni vplivi kvadrata, z zmanjševanjem pa stroški, povezani z rekurzivnim razcepom.

[3] Dovolj je, da je število n sodo. Ni nujno, da je potenca števila 2.