Домашна работа № 1 по "Дизайн и анализ на алгоритми"
за специалност "Компютърни науки", 2. курс, 1. поток — СУ, ФМИ,
летен семестър на 2017 / 2018 уч. г.

ИЗСЛЕДВАНЕ НА АЛГОРИТМИ

Задача 1. Нека $f(n)$ и $g(n)$ са асимптотично положителни функции, за които $f(n) \succ g(n)$. Докажете, че $f(n) - g(n) \asymp f(n)$.

Покажете с контрапример, че ако заменим строгото неравенство с нестрого, твърдението ще престане да бъде вярно.

Задача 2. Намерете порядъка на обратната функция на факториела:

$w(n)$ = най-голямото цяло положително число k, за което $k! \le n$.

Функцията $w(n)$ е определена за цели положителни стойности на n. Например $w(1) = 1,\ w(2) = w(3) = w(4) = w(5) = 2,\ w(6) = \cdots = w(23) = 3,\ w(24) = 4$.

Задача 3. Да се реши рекурентното уравнение $T(n) = T\left(\frac{n}{5}\right) + T\left(\frac{n}{8}\right) + n^3$.

Задача 4. Намерете максималната времева сложност на дадения алгоритъм.

```
Alg(A[1...n]: array of integers)  // масив от цели числа
S: stack
S ← empty stack                   // празен стек
for k ← 1 to n
   push(S, A[k])                  // добавяне на елемент към стека
   x ← 4
   while (S is not empty) and (x is even)
      x ← pop(S)                  // изваждане на елемент от стека
```

Точките за всяка задача са 25, а за цялото домашно — най-много 100.

Идентични решения се дисквалифицират!

РЕШЕНИЯ

Задача 1. $f(n) \succ g(n) \Rightarrow \lim\limits_{n\to\infty} \dfrac{g(n)}{f(n)} = 0$

$$\Rightarrow \lim_{n\to\infty} \frac{f(n)-g(n)}{f(n)} = \lim_{n\to\infty}\left(1-\frac{g(n)}{f(n)}\right) = 1 - 0 = 1 \Rightarrow f(n)-g(n) \asymp f(n).$$

20 точки

Ако асимптотичното неравенство е нестрого, то ще допуска възможността за асимптотично равенство на двете функции, тоест $g(n) \asymp f(n)$. Да вземем $f(n)=n^2+7n$ и $g(n)=n^2+2n$. Тогава $f(n)-g(n) = 5n \prec f(n)=n^2+7n$, следователно не е вярно, че $f(n)-g(n) \asymp f(n)$. **5 точки**

Задача 2. От определението на функцията w следва, че $w(n)\to\infty$ при $n\to\infty$ и че е изпълнено следното двойно неравенство:

$$w! \le n < (w+1)!.$$

Логаритмуваме:

$$w\log w \asymp \log(w!) \le \log n < \log((w+1)!) \asymp (w+1)\log(w+1) \asymp w\log w.$$

От неравенствата $w\log w \preceq \log n \preceq w\log w$ следва, че $\log n \asymp w\log w$.

Затова $w \asymp \dfrac{\log n}{\log w}$. **5 точки**

Развиваме това уравнение, но само с една стъпка:

$$w \asymp \frac{\log n}{\log\log n - \log\log w}.$$ **5 точки**

(Ако извършим повече стъпки, знаменателят ще се усложни.)

Прилагаме свойството от задача 1: тъй като $\log\log n \succ \log\log w$, то знаменателят се опростява до $\log\log n$, тоест

$$w(n) \asymp \frac{\log n}{\log\log n}.$$ **5 точки**

Това е отговорът на задачата — порядъкът на обратната функция на факториела.

Обаче в горните разсъждения използвахме без доказателство неравенството

$$\log \log n \succ \log \log w.$$

Доказателство: По-горе установихме, че $\log n \asymp w \log w$, следователно $\log \log n \asymp \log (w \log w) = \log w + \log \log w \asymp \log w \succ \log \log w$. **5 точки**

Остава да докажем неравенството

$$\log w \succ \log \log w,$$

което се използва два пъти в горната редица от асимптотични сравнения:

1) самостоятелно — в последната стъпка;
2) като основание за пренебрегване на събираемото от по-нисък порядък — в предпоследната стъпка.

Неравенството $\log w \succ \log \log w$ се доказва чрез граничен преход, в който се използва правилото на Лопитал:

$$\lim_{w \to \infty} \frac{\ln w}{\ln \ln w} = \left[\frac{\infty}{\infty}\right] = \lim_{w \to \infty} \frac{(\ln w)'}{(\ln \ln w)'} =$$

$$= \lim_{w \to \infty} \frac{\frac{1}{w}}{\frac{1}{\ln w} \cdot \frac{1}{w}} = \lim_{w \to \infty} \frac{1}{\frac{1}{\ln w}} = \lim_{w \to \infty} \ln w = \infty.$$ **5 точки**

Задача 3 може да се реши по индукция или чрез развиване на уравнението и изследване на полученото дърво на рекурсията. **25 точки**

Отговор: $T(n) \asymp n^3$.

Задача 4. Особеното тук е, че лесно можем да получим грешен отговор. За удобство нека отделим вътрешния цикъл в самостоятелен подалгоритъм.

```
ALG(A[1...n])
S: stack
S ← empty stack
for k ← 1 to n
   push(S, A[k])
   SUB_ALG(S)
```

```
SUB_ALG(S)
x ← 4
while (S is not empty) and (x is even)
   x ← pop(S)
```

Да означим времевите сложности в най-лошия случай, както следва:
$T_1(n)$ = сложността на `ALG`; $T_2(n)$ = сложността на `SUB_ALG`.

Най-лош случай за подалгоритъма `SUB_ALG` е, когато стекът S съдържа всички n числа от масива A и те са четни. Всяко число се вади от стека, докато той се опразни: операцията `pop` се изпълнява n пъти, затова $T_2(n) \asymp n$.

Най-лош случай за алгоритъма `ALG` е, когато всяко от извикванията на подалгоритъма `SUB_ALG` е възможно най-бавно. Затова сложността на `ALG` е равна на $T_1(n) \asymp n \,.\, T_2(n) \asymp n \,.\, n = n^2$, тоест $T_1(n) \asymp n^2$. **5 точки**

Тези разсъждения не са достатъчни, тъй като все още не сме проверили дали описаната ситуация е възможна. Отнапред не е ясно дали е възможно всяко от извикванията на `SUB_ALG` да бъде бавно. Затова полученият резултат е само горна граница за сложността на `ALG`. Точният запис е с неравенство: $T_1(n) \preceq n \,.\, T_2(n) \asymp n \,.\, n = n^2$, тоест $T_1(n) \preceq n^2$. **5 точки**

Това неравенство е вярно, обаче то не може да се приеме за отговор на задачата, защото търсим точния порядък на алгоритъма, т.е. равенство.

Равенството $T_1(n) \asymp n^2$ не е вярно, тъй като операцията `pop` не може да бъде изпълнена $\Theta(n^2)$ пъти: няма как да извадим от стека повече елементи, отколкото сме сложили в него, а сме сложили точно n елемента, понеже операцията `push` се изпълнява точно n пъти — по веднъж за всеки елемент на входния масив A. **5 точки**

И така, след като операцията `pop` се изпълнява не повече от n пъти, а операцията `push` — точно n пъти, то $T_1(n) \asymp n$. Това е вярната оценка за сложността на алгоритъма `ALG`. За него няма най-лош случай; по-точно, времето за изпълнението му е $\Theta(n)$ при всички входни данни. **10 точки**

Щом операцията `pop` се изпълнява не повече от n пъти общо за цялото изпълнение на алгоритъма `ALG`, а пък `ALG` вика `SUB_ALG` точно n пъти, то операцията `pop` се изпълнява средно не повече от веднъж при едно извикване на `SUB_ALG`. От друга страна, при всяко извикване на `SUB_ALG` се изпълняват поне три операции — присвояването, проверката за празен стек и проверката за четност. Затова средният брой операции на едно извикване на подалгоритъма `SUB_ALG` е константа: $\widetilde{T_2}(n) \asymp 1$. Функцията $\widetilde{T_2}(n)$ се нарича ***амортизирана сложност*** на подалгоритъма `SUB_ALG`. Тя показва средния брой операции, извършени от подалгоритъма при едно негово извикване от главния алгоритъм. Тя не е средна сложност, защото усредняването е по извикванията от главния алгоритъм, а не по входните данни на подалгоритъма.