Serii: CA, CD 5 noiembrie 2018

Test 1 – Analiza Algoritmilor

Timp de lucru: 90 de minute
Punctaj total: 18p

1. (3p) Fie mulțimile $A, B, C \subseteq \mathbb{N}$ despre care știm că:
   (1) $P_A \leq_T P_B$, unde P_A, P_B sunt programele care verifică apartenența unui element în mulțimea A, respectiv mulțimea B.
   (2) B este o mulțime recursivă.
   (3) C este o mulțime recursiv-enumerabilă.

Ce se poate spune despre decidabilitatea lui $P_{C\setminus A}$ (programul corespunzător mulțimii $C\setminus A$)? Justificați.

Rezolvare: Din (1) si (2) rezultă că A este recursivă (1p)

Cum A este recursivă, rezultă că $P_A(x) = \begin{cases} 1, & x \in A \\ 0, & altfel \end{cases}$. Din (3) rezultă că există un program generator al mulțimii C, notat Q_C. Scriem programul generator pentru $C\setminus A$ (arătăm că $C\setminus A$ este recursiv-enumerabilă):

```
while(1) {
        x = Q_C();
        if (P_A(x) == 0)
                return x;
}
```

Practic, se generează elemente din C, iar dacă acestea nu se află și în A sunt returnate.
Se observă că nu se poate afirma că $C\setminus A$ este recursivă deoarece, dacă ar fi recursivă, ar rezulta că $C = (C\setminus A) U\ A$ este recursivă (ceea ce nu este întotdeauna adevărat)
Notă barem: Se punctează cu 2p orice program corect care ajunge la aceeași concluzie (inclusiv un pseudocod scriu în cuvinte, care explică ce face programul)

2. (2p) Fie funcțiile $f, g: \mathbb{N} \to \mathbb{R}_+$ astfel încât $f \in \theta(g(n))$. În ce clasă de complexitate se află funcția $|f(n) - g(n)|$?

Rezolvare: $|f(n) - g(n)| \leq \max(f(n), g(n)) \in \theta(g(n))$, rezultă că $|f(n) - g(n)| \in O(g(n))$ (1p)

Pe de altă parte, avem $f \in \theta(g(n))$, deci există constantele c_1, c_2 astfel încât

$c_1 g(n) \leq f(n) \leq c_2 g(n) \Leftrightarrow (c_1 - 1)g(n) \leq f(n) - g(n) \leq (c_2 - 1)g(n)$ pentru n suficient de mare. În cazul în care $c_1 = 1$ sau $c_2 = 1$, aplicând modul ecuației de mai sus rezultă că $|f(n) - g(n)|$ este mărginită superior de o constantă înmulțită cu g(n) și inferior de 0, deci nu putem afirma că $|f(n) - g(n)| \in \theta(g(n))$ (1p)

Notă barem: Dacă se foloseau corect definițiile, dar se ajungea la rezultatul final $\theta(g(n))$ (de exemplu, aplicând greșit funcția modul inegalității de mai sus), se va puncta cu 1p.

Dacă se explica în cuvinte ultima parte a rezolvării sau se dădea un exemplu care arată că nu se poate spune că $|f(n) - g(n)| \in \theta(g(n))$ (de exemplu, f(n) = g(n) sau f(n) = n + 1 si g(n) = n, etc.), se vor primi 2p.

Se va puncta identic dacă rezultatul final este în funcție de f(n) și nu de g(n)

3. (2x2p) Calculați complexitatea următoarelor bucăți de cod:
   a) $for(int\ i = 1; i \leq n; i *= 2)$
      $for(int\ j = n; j > 0; j /= 4)$
         $for(int\ k = i; k \leq n; k++)$
            //număr constant c de operații
   b) $for(int\ i = 1; i < n; i += 3)$
      $for(int\ j = 2 * i; j \leq n; j *= 2)$
         //număr constant c de operații

Rezolvare: a) Numărăm de câte ori se intră în for-ul după k. Observăm că for-ul după j este independent de celelalte, deci va contribui la complexitatea finală cu θ($\log_4 n$). (0.5 p)

Pentru fiecare valoare a lui i, k ia n – i + 1 valori. (0.5p) Rezultă complexitatea

$(n-1+1) + (n-2+1) + (n-4+1) + \cdots + \left(n - 2^{\log_2 n} + 1\right) = n(\log_2 n + 1) - \left(1 + 2 + \cdots + 2^{\log_2 n}\right) + \log_2 n + 1 = n \log_2 n + n - (2^{\log_2 n + 1} - 1) + \log_2 n + 1 = n \log_2 n - n + \log_2 n + 2 \in \theta(n \log_2 n)$(1p).

Înmulțind și cu θ($\log_4 n$) menționat mai sus, rezultă complexitatea finală $\theta(nlog_2^2 n)$ (în rezultatul final am folosit și că baza logaritmului nu contează în clasa de complexitate)

b) Numărăm de câte ori se intră în for-ul după j. Pentru fiecare valoare a lui i, j ia $\log_2(\frac{n}{2i})$ valori. Deoarece i ia valori din 3 în 3, rezultă complexitatea:

$$\sum_{k=0}^{\frac{n-2}{3}} \log_2\left(\frac{n}{2(1+3k)}\right) = \frac{n-2}{3}\log_2 n - \frac{n-2}{3} - \sum_{k=0}^{\frac{n-2}{3}} \log_2(1+3k) =$$

$= \theta(n \log_2 n) - \sum_{k=0}^{\frac{n-2}{3}} \log_2(1+3k)$, deci complexitatea finală este $O(n \log_2 n)$.

Onservație: Se puncteaza orice soluție corectă care ajunge la un rezultat din care rezultă clar complexitatea $O(n \log_2 n)$ (orice rezultat de forma $c * n \log_2 n - f(n)$)

4. (2p) Rezolvați, folosind metoda substituției (inducție), următoarea recurență, după încadrarea ei în clasa de complexitate θ:
   $T(n) = 2T(n/4) + \sqrt{n}$

Rezolvare: Arătăm prin inducție că $T(n) \in \theta(\sqrt{n} * \log_2 n)$ (soluția poate fi gasită cu teorema Master, cazul 2)

Cazul de bază (0.5p): $T(1) = 1$ se consideră un caz neconvenabil, deoarece ar rezulta $0 \leq T(1) \leq 0$. Demonstrăm pentru n = 4, adică

$c_1\sqrt{4}\log_2 4 \leq \ T(4) \leq c_2\sqrt{4}\log_2 4 \Leftrightarrow \ 4c_1 \leq 2T(1) + 2 \leq 4c_2 \Leftrightarrow c_1 \leq 1 \leq c_2$

Evident, există constante cu această proprietate.

Pentru pasul de inducție, presupunem adevărat P(n/4) și arătăm P(n)

$P\left(\frac{n}{4}\right): c_1\sqrt{\frac{n}{4}}\log_2\frac{n}{4} \leq T\left(\frac{n}{4}\right) \leq c_2\sqrt{\frac{n}{4}}\log_2\frac{n}{4}$ (0.5p)

Înmulțim cu 2 și adunăm $\sqrt{n}$ și obținem

$c_1\sqrt{n}\log_2 n + \ \sqrt{n}(1 - 2c_1) \leq T(n) \leq \ c_2\sqrt{n}\log_2 n + \ \sqrt{n}(1 - 2c_2)$(0.5p)

Punem condițiile $c_1\sqrt{n}\log_2 n + \ \sqrt{n}(1 - 2c_1) \geq \ c_1\sqrt{n}\log_2 n$, $c_2\sqrt{n}\log_2 n + \sqrt{n}(1 - 2c_2) \leq \ c_2\sqrt{n}\log_2 n$, care sunt adevărate pentru $c_1 \leq \frac{1}{2}$ și $c_2 \geq \frac{1}{2}$ (0.5p). Se observă că aceste constante respectă cazul de bază, ceea ce încheie demonstrația.

5. (2x2p) Rezolvați următoarele recurențe prin orice metodă:
   a) $T(n) = 3T(n/4) + n\ log_2\, n$
   b) $T(n) = \sqrt{n}\, T(\sqrt{n}) + 100n$

Rezolvare: a) Demonstrăm că ne aflăm în cazul 3 al teoremei Master. Avem $f(n) = n\log_2 n \ \in \ \Omega(\mathrm{n}) = \ \Omega(\mathrm{n}^{\log_4 3 + \varepsilon})$, pentru $\varepsilon = 1 - \log_4 3$ (0.5p).

Mai avem de demonstrat că există o constantă c subunitară astfel încât $3f\left(\frac{n}{4}\right) \leq cf(n)$, pentru n suficient de mare. Inegalitatea devine $\frac{3n}{4}\log_2\frac{n}{4} \leq cnlog_2 n \Leftrightarrow$

$\frac{3}{4}\log_2 n - \frac{3}{2} \leq clog_2 n,$ ceea ce este adevărat pentru c = 3/4 . (1p)

Rezultă complexitatea finală $\theta(nlog_2 n)$ (0.5 p)

b) Folosim metoda iterației. Avem:

$$T(n) = \ \sqrt{n}T(\sqrt{n}) + 100n$$

$$\sqrt{n}T(\sqrt{n}) = n^{\frac{1}{2}}n^{\frac{1}{4}}T\left(n^{\frac{1}{4}}\right) + n^{\frac{1}{2}} * 100n^{\frac{1}{2}} = n^{\frac{3}{4}}T\left(n^{\frac{1}{4}}\right) + 100n$$

$$n^{\frac{3}{4}}T\left(n^{\frac{1}{4}}\right) = \ n^{\frac{3}{4}}n^{\frac{1}{8}}T\left(n^{\frac{1}{8}}\right) + n^{\frac{3}{4}} * 100n^{\frac{1}{4}} = \ n^{\frac{7}{8}}T\left(n^{\frac{1}{8}}\right) + 100n$$

...

$$n^{\frac{1}{h}}T\left(n^{\frac{1}{h}}\right) = 100n$$

unde $h = \log_2\log_2 n$. Însumând toate ecuațiile, rezultă:

$$T(n) = (h + 1) * 100 * n\ \epsilon\ \theta(nlog_2\log_2 n)$$

Observație: Se punctează orice soluție corectă care ajunge la rezultat (de exemplu, se putea împărți ecuația inițială la n și să se noteze S(n) = T(n) / n)

6. (3p) Fie problema $check\_even$ exprimată în felul următor:

$$check\_even(P) = \begin{cases} 1, & \forall x, \quad P(x) = 1 \Leftrightarrow x\ par \\ 0, & altfel \end{cases}$$

$check\_even$ decide, așadar, dacă un program P întoarce 1 pentru toate intrările lui pare și numai pentru acestea.

Arătați că problema $check\_even$ **nu** este decidabilă folosind o reducere Turing.

Rezolvare: Demonstrăm că problema opririi (PO) se reduce Turing la check_even. Trebuie să transformăm orice intrare a lui PO (adică o pereche (P, w)) într-o intrare a lui check_even (adică un program, notat în continuare P') cu $PO(P, w) = 1 \Leftrightarrow check\_even(P') = 1.$

Scriem programul P' astfel (1.5p pentru orice program corect pentru care se poate ajunge la concluzie):

$P'(x): \{$
$P(w);$
$return\ 1 - x\ \%\ 2;$
$\}$

Trebuie să demonstrăm acum echivalența.
$PO(P, w) = 1 \Leftrightarrow P(w) se\ termin$ă $\Leftrightarrow P' returnea z$ă $1 - x\ \%\ 2$ pentru orice x (adică 1 pentru x par și 0 pentru x impar) $\Leftrightarrow$ check_even(P') = 1. (1.5p)
Observație: Dacă x este impar, iar P' returnează 0, check_even are soluție (P' este instanță-da) fiindcă se verifică proprietatea din ipoteză (prima ramură) în mod corect.