AA Problem Pool

8 ianuarie 2012

Cuprins

1 Notaţii de complexitate, recurenţe

Notă: În absenţa altor precizări, se consideră că soluţiile recurenţelor trebuie furnizate utilizând aproximări strânse (notaţia Θ).

1. Treceti in dreptul fiecarei intrari din tabelul de mai jos cea mai precisa relatie dintre f si g, alegand dintre $O, \Theta, \Omega, o, \omega$:

$f(n)$	$g(n)$	$f(n) = \ldots (g(n))$
$n^3 + 3n + 1$	n^4	
$\lg n$	n	
$n2^n$	3^n	
n	$\lg^5 n$	
$\lg n$	$\lg n^2$	
$100n + \lg n$	$n + \lg^2 n$	

   Justificati fiecare alegere facuta.

2. Creati o ordonare a functiilor urmatoare astfel incat, daca $f(n) \in O(g(n))$, atunci $f(n)$ sa apara inaintea lui $g(n)$ in ordonare. Mentionati daca exista functii care au aceeasi crestere asimptotica. Justificati ordonarea propusa.

   $n^2,\ n \lg n,\ n^3 + \lg n,\ \sqrt{n},\ n^2 + 2n \lg n,\ \lg \lg n,\ 17 \lg n,\ 10n^{3/2},\ n^5 - n^4 + 2n,\ 5n^2 \lg \lg n,\ 3n^2 + n^3 \lg n,\ n + 6 \lg n$

3. Rezolvati urmatoarele recurente folosind metoda iteratiei:

- $A(n) = 2A(n/4) + \sqrt{n}$
- $B(n) = 3B(n/3) + n^2$

4. Fie recurenţa $T(n) = T(n/2) + T(n/4) + \Theta(n^2)$. Identificaţi o aproximare strânsă a soluţiei ecuaţiei de mai sus, utilizând metoda arborilor de recurenţă. Apoi, verificaţi soluţia găsită, folosind metoda substituţiei.

5. Rezolvaţi recurenţele:

   (a) $T(n) = T(\sqrt{n}) + 1$

   (b) $T(n) = \sqrt{n}\, T(\sqrt{n}) + 2011n$

   (c) $T(n) = T(n/2 + \sqrt{n}) + n$

6. Rezolvaţi recurenţa $T(n) = T(n/3) + T(n/6) + \Theta(n^{\sqrt{\lg n}})$.

   Indicaţie: Mărginiţi inferior şi superior membrul drept al egalităţii de mai sus, pentru a obţine recurenţe rezolvabile prin metoda master.

7. Se considera un vector cu n numere reale $X[0 \ldots n-1]$. Notam cu $RMQ_X(l, r)$ (Range Minimum Query [1]) urmatoarea problema: *Sa se gaseasca cea mai mica valoare din subvectorul* $X[l, r]$*, cu* $0 \leq l < r \leq n - 1$.

   O instanta a problemei $RMQ_X(l, r)$ poate fi rezolvata folosind un algoritm traditional de calculare a minimului pentru vectorul $X[l, r]$.

   (a) Care este complexitatea rezolvarii $RMQ_X(l, r)$ in aceasta situatie ?

   (b) Se observa cu usurinta ca, pentru o secventa de m intrebari $RMQ_X(l, r)$, abordarea de mai sus devine ineficienta. Putem reduce complexitatea rezolvarii celor m intrebari, adaugand o etapa de **preprocesare** a vectorului X. Scrieti un algoritm de preprocesare a vectorului X, care asigura un timp $O(1)$ pentru a raspunde la intrebari $RMQ_X(l, r)$. Care este complexitatea acestuia ? (Puteti folosi orice tip de structuri de date pentru memorarea valorilor minime).

   (c) Algoritmul de **preprocesare** de la punctul anterior va avea o complexitate spatiala mare. Pentru a o reduce, fie urmatoarea alternativa: impartim vectorul X in n/k subvectori de lungime k (la care se adauga un posibil ultim subvector de dimensiune $< k$). Pentru fiecare dintre acesti subectori, vom calcula minimul si il vom retine intr-un vector Max. Scrieti un algoritm de **preprocesare**, care construieste vectorul Max. Pe baza acestuia, scrieti un algoritm care rezolva $RMQ_X(l, r)$.

   (d) Analizati complexitatea rezolvarii a m intrebari $RMQ_X(l, r)$, pe baza algoritmului de mai sus.

   (e) Cum influenteaza valoarea k complexitatea rezolvarii $RMQ_X(l, r)$? Identificati valorile k pentru cazurile cele mai favorabile si defavorabile.

[1] `http://en.wikipedia.org/wiki/Range_Minimum_Query`

2 Analiză amortizată

1. Se considera o implementare a unei cozi FIFO (First In First Out), folosind doua stive S_1 si S_2. Stiva are operatiile PUSH si POP, iar costul fiecarei operatii este 1. Coada are operatiile ENQUEUE si DEQUEUE, implementate astfel:

```
ENQUEUE(x) {
        do PUSH(x) on S1
}
DEQUEUE() {
        if (S2 is empty) {
                POP() all elements from S1 and PUSH them in S2
        }
        do POP() on S2 and return the result
}
```

   (a) Exemplificati continutul cozii FIFO (si implicit al stivelor) pentru o secventa oarecare (aleasa de voi) de operatii ENQUEUE si DEQUEUE;

   (b) Identificati costul mediu pentru o secventa de operatii ENQUEUE si DEQUEUE, folosind metoda agregarii;

   (c) Identificati costul amortizat pentru operatiile ENQUEUE si DEQUEUE folosind metoda creditelor;

   (d) Identificati o functie de potential pentru coada FIFO astfel incat costurile amortizate care rezulta pe baza acesteia sa fie cele identificate la punctul anterior.

2. Se considera o stiva implementata folosind un vector. Operatiile definite pentru stiva sunt push si pop. Totusi, cand se adauga un element in stiva (push) si vectorul este plin, trebuie sa se aloce un nou vector si sa se copieze elementele din vectorul vechi in cel nou, iar apoi se va folosi vectorul nou pe post de stiva.

   (a) Care este complexitatea operatiilor push si pop in cazul cel mai defavorabil ? Justificati.

   (b) Daca dimensiunea vectorului se incrementeaza cu 1 atunci cand este necesar un vector mai mare, care este costul total pentru n operatii si care este costul mediu (amortizat) al unei operatii pe stiva, folosind metoda agregarii ?

   (c) Daca dimensiunea vectorului se dubleaza atunci cand este necesar un vector mai mare, care este costul total pentru n operatii si care este costul mediu (amortizat) al unei operatii pe stiva, folosind metoda agregarii ?

3 Invarianţi la ciclare

1. Demonstraţi corectitudinea algoritmului *bubble sort*, utilizând invarianţi la ciclare.

Algoritmul 1 BubbleSort(A, n)

1: **for** $i = 1$ to $n - 1$ **do**
2: **for** $j = n$ downto $i + 1$ **do**
3: **if** $A[j] < A[j - 1]$ **then**
4: $A[j] \leftrightarrow A[j - 1]$
5: **end if**
6: **end for**
7: **end for**

2. Algoritmul lui Euclid primeste doua valori intregi A, B si calculeaza cel mai mare divizor comun:

```
b = B, a = A, r = B
while (b != 0) {
        r = a mod b
        a = b
        b = r
}
```

Demonstrati corectitudinea Algoritmului lui Euclid folosind invarianti la ciclare.

4 Inducţie structurală

1. Fie tipul de date $BTree$ definit prin constructorii:

$$BTEmpty :\rightarrow BTree$$

.

$$BTNode : BTree \times T \times BTree$$

si definitiile: $flattenTree(t) : BTree \rightarrow List$

- $(F1) flattenTree(BTEmpty) = []$
- $(F2) flattenTree(BTNode(left, i, right)) = flattenTree(left) ++ [i] ++ flattenTree(right)$

$numBTElem(t) : BTree \rightarrow \mathbb{N}$

- $(N1) numBTElem(BTEmpty) = 0$
- $(N2) numBTElem(BTNode(left, i, right)) = 1 + numBTElem(left) + numBTElem(right)$

$length(l) : LIST \rightarrow \mathbb{N}$

- $(L1) length([]) = 0$
- $(L2) length(h : t) = 1 + length(t)$

Constructorul $h : t$ sta pentru $cons(h, t)$, pentru tipul LIST, [] reprezinta un constructor nular (lista vida), iar $[a]$ sta pentru o lista cu un element.

Operatorul $++$ se refera la concatenarea a doua liste. El respecta urmatoarea proprietate: Fie L_1, L_2 doua liste. Atunci $length(L_1 ++ L_2) = length(L_1) + length(L_2)$.

Sa se demonstreze ca:

$$\forall T \in BTRree.numBTElem(t) = length(flattenTree(t))$$

2. Fie limbajul restricţionat al logicii cu predicate de ordinul I, cu următoarele categorii de expresii:

   - **Termeni**:
     - **Constante** şi **Variabile**: Dacă x este o constantă sau o variabilă, atunci x este un termen;
     - **Aplicaţii de funcţii**: Dacă f este o funcţie şi $t_1, \ldots, t_n$ sunt termeni, atunci $f(t_1, \ldots, t_n)$ este un termen. Exemple:
       - $succesor(4)$: întoarce succesorul lui 4, şi anume 5;
       - $+(2, x)$: semnifică aplicaţia funcţiei de adunare asupra numerelor 2 şi x, şi, totodată, suma lor;
   - **Atomi**: Dacă P este un predicat şi $t_1, \ldots, t_n$ sunt termeni, atunci $P(t_1, \ldots, t_n)$ este un atom. Exemple:
     - $Impar(3)$;
     - $Varsta(ion, 20)$;
     - $=(+(2, 3), 5)$;
   - **Formule**: Dacă x este o variabilă, A un atom, iar F şi G formule, atunci următoarele sunt, de asemenea, formule:
     - **Fals**, **adevărat**: $\bot$, $\top$;
     - **Atomi**: A;
     - **Negaţii**: $\neg F$;
     - **Conjuncţii**: $(F \wedge G)$;
     - **Cuantificări**: $\forall x.F$.

   Formulele au asociate valori de adevăr. De exemplu, fie formula

$$F \equiv \forall \underbrace{x}_{x_1} . \underbrace{(\geq (\underbrace{x}_{x_2}, \underbrace{s}_{s_1}) \wedge \exists \underbrace{s}_{s_2} . = (+(\underbrace{x}_{x_3}, 1), \underbrace{s}_{s_3}))}_{G} .$$

   Notă: F conţine, pentru o lizibilitate sporită, cuantificatorul existenţial, care nu este inclus în limbajul restricţionat descris. Cu toate acestea, ea poate fi rescrisă echivalent, astfel:

$$F \equiv \forall x.(\geq (x, s) \wedge \neg \forall s. \neq (+(x, 1), s)).$$

   Formula F afirmă că, oricum am alege un număr x, acesta este mai mare sau egal cu un număr fixat, s, şi există un alt număr, numit tot s, care constituie succesorul acestuia. De remarcat că apariţia s_1, pe de o parte, şi apariţiile s_2 şi s_3, pe de alta, sunt independente, referindu-se, de fapt, la valori diferite: s_1 este aceeaşi pentru toate valorile lui x, în timp ce s_2

şi s_3 pot depinde de valorile particulare. Mai mult, am putea redenumi aparitiile s_2 şi s_3, de exemplu, prin t, păstrând semnificaţia formulei. Nu acelaşi lucru s-ar putea spune şi despre aparitia s_1. Aceasta ar putea avea o semnificaţie globală, fiind legată la o anumită valoare. În domeniul numerelor naturale, F este adevărată dacă legăm variabila s (corespunzând aparitiei s_1) la valoarea 0, şi falsă altfel. În domeniul numerelor întregi, F este întotdeauna falsă, deoarece ar trebui să legăm variabila s la cel mai mic număr întreg, care nu există.

O *aparitie* x' a unei variabile x se numeşte *legată* într-o formulă, dacă x' apare în oricare din situaţiile:

- $\forall x'. \ldots$, deci imediat după cuantificator;
- $\forall x. \ldots x' \ldots$, deci într-o formulă în care x este cuantificată.

O *variabilă* se numeşte *legată* într-o formulă, dacă toate aparitiile ei sunt legate în acea formulă. Altfel, se numeşte *liberă*. Exemple:

- În formula F, toate aparitiile lui x sunt legate, din moment ce formula începe cu $\forall x$. Prin urmare, x este legată în F;
- Prin opoziţie, în formula G, toate aparitiile lui x sunt libere, deoarece x nu este cuantificată. Prin urmare, x este liberă în G;
- În schimb, în formulele F şi G, aparitia s_1 este liberă, în timp ce aparitiile s_2 şi s_3 sunt legate. Prin urmare, s este liberă în F şi G.

Fie operatorii FV (*free variables*) şi BV (*bound variables*), astfel încât, pentru formula F, $FV(F)$ întoarce multimea variabilelor libere din formula F, iar $BV(F)$ multimea variabilelor legate din aceeaşi formulă.

Cerinţe:

(a) Identificaţi tipurile de date din descrierea de mai sus, alături de constructorii de bază, precizând signatura şi felul acestora (nulari, externi, interni);

(b) Definiţi signaturile şi axiomele corespunzătoare operatorilor FV şi BV;

(c) Demonstraţi, prin inductie structurală, că, pentru orice formulă F, $FV(F) \cap BV(F) = \emptyset$.

5 Nedeterminism

1. Demonstraţi că problema 2-SAT este în **P**:

   2-*SAT* Este o formulă FNC, în care fiecare termen conţine doi literali, satisfiabilă?

2. Fie problemele de mai jos. Sunt ele în **P**?

   FNC-tautologie Este o formulă FNC adevărată pentru orice legare a variabilelor sale?

   FND-tautologie Similar, unde o fromulă FND (*forma normală disjunctivă*) conţine SAU între termeni şi $ŞI$ în interiorul termenilor.

3. Scrieţi un algoritm nedeterminist care verifică dacă, într-un graf neorientat, există un drum simplu între două noduri fixate, de cost cel puţin k. Care este complexitatea angelică a algoritmului?

4. Scrieţi un algoritm nedeterminist care verifică dacă o mulţime de numere poate fi partiţionată în două submulţimi, având sumele elementelor egale. Care este complexitatea angelică a algoritmului?

5. Pentru oricare din cei doi algoritmi de mai sus, scrieţi un algoritm determinist echivalent şi analizaţi complexitatea lui.

6. Demonstrati ca urmatoarele probleme sunt NP-complete:

   - *k-Vertex-Cover*;
   - *Set-Cover*;
   - *Independent-Set*

   In cadrul demonstratiei, se stie ca problema *3-SAT* este NP-completa. Nu se cunosc (si nu pot fi folosite) alte rezultate. Demonstratiile pot fi facute in orice ordine, iar rezultatele pot fi folosite in demonstratiile ulterioare. (Spre exemplu, dupa demonstratia NP-completitudinii lui *Set-Cover*, se pot construi reduceri plecand de la aceasta problema).

6 Decidabilitate

1. Demonstrati ca problema terminarii unui program care primeste, ca intrare, propria sa codificare, este semidecidabila.

2. Demonstraţi că reuniunea şi intersecţia a două mulţimi recursive sunt recursive.

3. Demonstraţi că reuniunea şi intersecţia a două mulţimi recursiv-numărabile sunt recursiv-numărabile.