Algoritmi şi structuri de date (I). Seminar 4: Analiza eficienţei algoritmilor. Identificarea celui mai favorabil şi a celui mai defavorabil caz. Estimarea timpului de execuţie prin contorizarea operaţiei dominante. Analiza în cazul mediu. Stabilirea ordinului de complexitate.

Problema 1 Scrieţi un algoritm care determină numărul de inversiuni ale unei permutări (pentru o permutare reprezentată printr-un tablou $a[1..n]$ o inversiune este o pereche de indici (i, j) cu proprietăţile $i, j \in \{1, \ldots, n\}$, $i < j$ şi $a[i] > a[j]$. Determinaţi numărul de operaţii de comparare (asupra elementelor tabloului a) efectuate de către algoritm şi stabiliţi ordinul de complexitate al algoritmului.

Problema 2 Se consideră algoritmul de mai jos care verifică dacă un tablou este ordonat crescător. Analizaţi eficienţa atât în cazurile extreme (cel mai favorabil, cel mai defavorabil) cât şi în cel mediu.

`ordonat`$(a[1..n])$

1: $r \leftarrow$ **true**
2: $i \leftarrow 0$
3: **while** $(i < n-1)$ **and** $(r =$**true**$)$
   do
4: $i \leftarrow i + 1$
5: **if** $a[i] > a[i+1]$ **then**
6: $r \leftarrow$ **false**
7: **end if**
8: **end while**
9: **return** r

Indicaţie. Pentru analiza în cazul mediu se poate porni de la ipoteza că există n clase distincte de date de intrare. Clasa C_i ($i \in \{1, \ldots, n-1\}$) corespunde tablourilor pentru care prima pereche de valori consecutive care încalcă condiţia de ordonare crescătoare este $(a[i], a[i+1])$, iar clasa C_n corespunde tablourilor ordonate.

Problema 3 Scrieţi un algoritm pentru a verifica dacă elementele unui tablou (cu valori întregi) sunt distincte sau nu. Estimaţi timpul de execuţie considerând comparaţia între elementele tabloului drept operaţie dominantă şi stabiliţi ordinul de complexitate al algoritmului.

Problema 4 Scrieţi un algoritm pentru a verifica dacă elementele unui tablou (cu valori întregi) sunt toate identice sau nu. Estimaţi timpul de execuţie considerând comparaţia între elementele tabloului drept operaţie dominantă şi stabiliţi ordinul de complexitate al algoritmului.

Problema 5 Se consideră un şir de valori întregi (pozitive şi negative, astfel încât cel puţin un element este pozitiv). Să se determine suma maximă a elementelor unei secvenţe a şirului (succesiune de elemente consecutive). De exemplu pentru şirul $(1, -2, 1, 5, -2, 4, 1, -3)$ suma maximă e 9 corespunzătoare subsecvenţei $(1, 5, -2, 4, 1)$. (a) Propuneţi un algoritm de complexitate $\mathcal{O}(n^2)$. (b) Propuneţi un algoritm de complexitate $\mathcal{O}(n)$.

Indicaţie. (a) Pentru fiecare pereche de indici (i, j) cu $i, j \in \{1, \ldots, n\}$ şi $i \leq j$ se calculează suma elementelor din subtabloul $a[i..j]$ şi se reţine cea maximă. (b) Se calculează succesiv suma elementelor din tablou pornind de la primul element pozitiv şi se reiniţiază calculul în momentul în care suma devine negativă (în acest caz se reiniţializează cu 0 variabila în care se colectează suma). La fiecare actualizare a variabilei în care se reţine suma se verifică dacă noua valoare este mai mare decât cea mai mare valoare a sumei determinată până la momentul respectiv, iar în caz afirmativ se reţine această nouă valoare.

Problema 6 Să se estimeze timpul de execuţie al unui algoritm care generează toate numerele prime mai mici decât o valoare dată n ($n \geq 2$).

Indicaţie. Se vor analiza două variante de rezolvare: (a) parcurgerea tuturor valorilor cuprinse între 2 şi n şi reţinerea celor prime; (b) algoritmul lui Eratostene.

(a) Presupunem că algoritmul `prim(i)` returnează **true** dacă n este prim şi **false** în caz contrar (analizând divizorii potenţiali dintre 2 şi $\lfloor\sqrt{i}\rfloor$).

(b) *Algoritmul lui Eratostene.* Se porneşte de la tabloul $a[2..n]$ iniţializat cu valorile naturale cuprinse între 2 şi n şi se marchează succesiv toate valorile ce sunt multipli ai unor valori mai mici. Elementele rămase nemarcate sunt prime.

Probleme suplimentare/teme

1. Se consideră o matrice A cu m linii şi n coloane şi o valoare x. Scrieţi un algoritm care verifică dacă valoarea x este prezentă sau nu în matrice. Stabiliţi ordinul de complexitate al algoritmului considerând comparaţia cu elementele matricii drept operaţie dominantă.

2. Se consideră un tablou $x[1..n-1]$ care conţine valori distincte din mulţimea $\{1, 2, \ldots, n\}$. Propuneţi un algoritm de complexitate liniară care să identifice valoarea din $\{1, 2, \ldots, n\}$ care nu este prezentă în tabloul x.

   Observaţie. Incercaţi să rezolvaţi problema folosind spaţiu de memorie auxiliar de dimensiune $\mathcal{O}(1)$ (aceasta înseamnă să nu folosiţi vectori auxiliari).

3. Se consideră un tablou $x[1..n]$ ordonat crescător cu elemente care nu sunt neapărat distincte. Să se transforme tabloul x astfel încât pe primele k poziţii să se afle elementele distincte în ordine crescătoare. De exemplu pornind de la $[1, 2, 2, 4, 4, 4, 7, 8, 9, 9, 9, 9]$ se obţine tabloul ce conţine pe primele $k = 6$ poziţii valorile: $[1, 2, 4, 7, 8, 9]$ (restul elementelor nu interesează ce valori au). Stabiliţi ordinul de complexitate al algoritmului propus.

4. Se consideră un tablou $x[1..n]$ şi o valoare $x0$ care este prezentă (pe o singură poziţie) în tablou. Presupunem că avem un algoritm de căutare care verifică elementele tabloului într-o ordine aleatoare (dar verifică fiecare element o singură dată). Procedura este similară celei în care avem într-o cutie $n-1$ bile negre şi o singură bilă albă şi efectuăm extrageri (fără a pune bila extrasă înapoi) până la extragerea bilei albe. Estimaţi numărul mediu de comparaţii (sau extrageri de bile) până la identificarea valorii căutate (extragerea bilei albe).

   Indicaţie. Se presupune că elementele (bilele) disponibile vor fi selectate cu aceeaşi probabilitate.

5. Se consideră un şir de caractere $s[1..n]$ şi un şablon (subşir de caractere) $t[1..m]$ (cu $m < n$). Scrieţi un algoritm care verifică dacă şablonul t este sau nu prezent în s şi stabiliţi ordinul de complexitate al algoritmului (considerând comparaţia dintre elementele şirului şi cele ale şablonului ca operaţie dominantă).

   Indicaţie. Cel mai simplu algoritm (nu şi cel mai eficient) se bazează pe parcurgerea şirului s de la poziţia $i = 1$ până la poziţia $i = n - m$ şi compararea element cu element al lui $s[i..i+m-1]$ cu $t[1..m]$.

6. Se consideră un tablou de valori întregi $x[1..n]$ şi o valoare dată, s. Să se verifice dacă există cel puţin doi indici i şi j (nu neapărat distincţi) care au proprietatea că $x[i] + x[j] = s$. Analizaţi complexitatea algoritmului propus.