Tema 2: Implementarea şi analiza complexităţii algoritmilor recursivi. Tehnici de reducere şi divizare. Tehnica greedy. Tehnica programării dinamice.

Termen: 11.01.2019

Submiterea temelor: Se uploadează prin Classroom (cf indicaţiilor primite de la cadrul didactic coordonator de laborator) o arhiva de tip *.zip având numele construit pe baza regulii: NumeStudent_seria_subgrupa_Tema2.zip (de exemplu `AdamEva_IR_sgr3_Tema3.zip`).

Arhiva trebuie să conţină următoarele fişiere:

- Un fişier în format PDF sau DOC care să conţină soluţiile propuse (răspunsuri la întrebări, algoritmi descrişi în pseudocod, explicaţii etc.)
- Câte un fişier cu codul sursă Python corespunzător implementărilor solicitate în enunţ denumit NumeStudent_seria_subgrupa_Tema2_Problema (de exemplu `AdamEva_IR_sgr3_Tema2_Pb1b.py`)

Punctaj total: 100p (+ bonus max 20p)

1. Se consideră următorii algoritmi recursivi:

```
1: alg1(a[0..n − 1])
2: if n == 1 then
3:    return a[0]
4: else
5:    temp← alg1(a[0..n − 2])
6:    if temp≤ a[n − 1] then
7:       return temp
8:    else
9:       return a[n − 1]
10:   end if
11: end if
```

```
1: alg2(k)
2: if k == 0 then
3:    print a[0..n − 1]
4: else
5:    for i ← 0, k do
6:       alg2(k − 1)
7:       if k MOD 2 == 0 then
8:          a[0] ↔ a[k − 1]
9:       else
10:         a[i] ↔ a[k − 1]
11:      end if
12:   end for
13: end if
```

   (a)(10p) Implementaţi `alg1` în Python, stabiliţi ce returnează algoritmul, scrieţi relaţia de recurenţă corespunzătoare numărului de comparaţii executate şi stabiliţi ordinul de complexitate al algoritmului.

   (b)(10p) Implementaţi `alg2` în Python utilizând tabloul $a[0..n-1]$ ca variabilă globală iniţializată cu $[0, 1, 2, ..., n-1]$, stabiliţi ce afişează algoritmul când este apelat ca `alg2`$(n-1)$, scrieţi relaţia de recurenţă corespunzătoare numărului de interschimbări executate şi stabiliţi ordinul de complexitate al algoritmului (în raport cu n).

2. Se consideră o matrice $A[1..m][1..n]$ având elementele de pe fiecare linie şi elementele de pe fiecare coloană ordonate crescător. Se pune problema verificării prezenţei unei valori v în matrice, folosind un număr cât mai mic de comparaţii.

   (a) (10p) Propuneţi un algoritm bazat pe tehnica divizării (prin extinderea ideii de la căutarea binară) şi stabiliţi ordinul de complexitate al algoritmului propus.

   (b) (10p) Descrieţi şi implementaţi algoritmul bazat pe următoarea idee:

- se porneşte căutarea din colţul dreapta sus al matricii ($i = 1$, $j = n$)
- dacă $v = A[i, j]$ atunci elementul a fost găsit şi se returnează True; dacă $v < A[i, j]$ atunci se continuă căutarea în stânga (se micşorează valoarea lui j); dacă $v > A[i, j]$ atunci se continuă căutarea în jos (se măreşte valoarea lui i); în cazul în care nu se mai poate continua căutarea (s-a ajuns la $i = m$ sau $j = 1$) atunci se returnează False.

Este corect algoritmul? Argumentaţi. Stabiliţi ordinul de complexitate.

3. (a)(10p) Se consideră un tablou $x[0..n-1]$ ordonat descrescător şi o valoare v. Propuneţi un algoritm de complexitate $\mathcal{O}(\log n)$ care determină indicele poziţiei pe care trebuie inserată valoarea v în cadrul tabloului astfel încât acesta să rămână ordonat descrescător. Justificaţi faptul că algoritmul are ordinul de complexitate cerut.

   (b)(5p) Modificaţi algoritmul de sortare descrescătoare prin inserţie astfel încât la fiecare etapă identificarea poziţiei de inserţie să se facă folosind algoritmul anterior. Stabiliţi ordinul de complexitate al noului algoritm de inserţie în raport cu numărul de comparaţii efectuate (asupra elementelor tabloului) precum şi în raport cu numărul de transferuri de elemente.

4. Se consideră un set de n locaţii între care există căi de acces având asociate diferite valori de câştig dacă sunt utilizate (dacă între două locaţii nu există cale de acces atunci câştigul asociat este 0). Câştigurile sunt stocate într-o matrice $C[1..n, 1..n]$ ($C[i, j] = C[j, i]$ reprezintă câştigul asociat căii dintre locaţiile i şi j). Se pune problema selectării a $n - 1$ căi directe de acces astfel încât orice locaţie să poată fi accesată din orice altă locaţie iar câştigul total asociat căilor selectate să fie maxim.
   Indicaţie. Problema este înrudită cu cea a determinării unui arbore minim de acoperire şi poate fi rezolvată folosind o idee similară celei din algoritmul lui Prim: se porneşte de la una dintre locaţii (selectată arbitrar) şi se adaugă succesiv calea de acces care are câştigul cel mai mare şi conectează una din locaţiile deja selectate cu o locaţie încă neselectată. Pentru detalii privind algoritmul lui Prim se poate consulta https://www.geeksforgeeks.org/prims-minimum-spanning-tree-mst-greedy-algo-5/

   (a)(10p) Descrieţi algoritmul în pseudocod şi testaţi implementarea în Python. Pentru testare se poate considera matricea de costuri: $c = [[0, 3, 0, 0, 6, 5], [3, 0, 1, 0, 0, 4], [0, 1, 0, 6, 0, 4], [0, 0, 6, 0, 8, 5], [6, 0, 0, 8, 0, 2], [5, 4, 4, 5, 2, 0]]$.

   (b)(5p) Stabiliţi ordinul de complexitate al algoritmului propus.

5. Secvenţele ADN sunt succesiuni de simboluri "A", "C", "G", "T" corespunzătoare celor patru tipuri de nucleotide: "adenină", "citozină", "guanină" şi "timină". Căutarea în bazele de date ce conţin secvenţe ADN se bazează pe alinierea secvenţelor prin maximizarea unui scor de potrivire. Două secvenţe (care iniţial pot fi de lungimi diferite) se consideră aliniate dacă prin introducerea unor spaţii (gap-uri) ajung să aibă aceeaşi lungime, astfel că fiecare element din prima secvenţă (nucleotidă sau gap) este pus în corespondenţă cu un element din a doua secvenţă (nucleotidă sau gap). Scorul unei alinieri se determină calculând suma scorurilor de potrivire ale elementelor corespondente din cele două secvenţe. O aliniere se consideră optimă dacă scorul său este maxim.

   Exemplu. Considerăm secvenţele "CGTTA" şi "AGTA" şi scorurile de potrivire: $+2$ (nucleotide identice aliniate), -2 (nucleotide diferite aliniate) şi -1 (nucleotidă aliniată cu un gap). In acest caz alinierea optimă (de scor maxim) este:
   AG-TA

CGTTA
cu scorul 3 ($= -2 + 2 - 1 + 2 + 2$).

(a)(10p) Scrieţi relaţia de recurenţă care permite calculul scorului corespunzător unei alinieri optime.

(b)(10p) Descrieţi în pseudocod şi implementaţi în Python algoritmul pentru calculul scorului corespunzător unei alinieri optime (folosind scorurile de potrivire între elemente specificate în exemplul de mai sus). Nu e necesară construirea alinierii.

Indicaţie. Se poate utiliza ideea, bazată pe tehnica programării dinamice, de la calculul distanţei de editare cu deosebirea că pentru elementele identice se adaugă scorul de potrivire, pentru cele diferite se adaugă scorul de nepotrivire iar în cazul ştergerii sau inserţiei se adaugă valoarea scorului corespunzător alinierii cu un gap. In plus, spre deosebire de calculul distanţei de editare unde se urmăreşte minimizarea numărului de operaţii în acest caz se urmăreşte maximizarea scorului de aliniere.

6. (10p) Se consideră trei tablouri ordonate crescător ($a[1..m]$, $b[1..n]$ şi $c[1..p]$). Propuneţi un algoritm de complexitate $\mathcal{O}(m+n+p)$ care construieşte tabloul d ordonat crescător ce conţine elementele comune din cele trei tablouri. Implementaţi algoritmul în Python şi justificaţi că are ordinul de complexitate cerut. *Indicaţie.* Se poate utiliza tehnica interclasării.

7. (Problemă suplimentară)(20p) *Generarea codului Gray.* Codul Gray este o variantă de reprezentare binară caracterizată prin faptul că valori consecutive au asociate secvenţe binare care diferă într-o singură poziţie. De exemplu pentru $n = 3$ codificarea Gray este: $0 : (0, 0, 0)$; $1 : (0, 0, 1)$; $2 : (0, 1, 1)$; $3 : (0, 1, 0)$; $4 : (1, 1, 0)$; $5 : (1, 1, 1)$; $6 : (1, 0, 1)$; $7 : (1, 0, 0)$. Codul poartă numele unui cercetător (Frank Gray) de la AT&T Bell Laboratories care l-a utilizat pentru a minimiza efectul erorilor de transmitere a semnalelor digitale.

   (a) Propuneţi un algoritm iterativ prin care codul Gray se obţine pornind de la codificarea clasică în baza 2. De exemplu pentru $n = 3$ transformarea constă în: $(0, 0, 0) \to (0, 0, 0)$; $(0, 0, 1) \to (0, 0, 1)$; $(0, 1, 0) \to (0, 1, 1)$; $(0, 1, 1) \to (0, 1, 0)$; $(1, 0, 0) \to (1, 1, 0)$; $(1, 0, 1) \to (1, 1, 1)$; $(1, 1, 0) \to (1, 0, 1)$; $(1, 1, 1) \to (1, 0, 0)$;

   (b) Propuneţi un algoritm recursiv care generează codul Gray de ordin n direct, fără a folosi reprezentarea în baza 2 a valorilor din $\{0, 1, \ldots, 2^{n-1}\}$.