Tema 2: Analiza complexităţii algoritmilor. Algoritmi de căutare şi sortare. Aplicaţii ale tehnicilor de proiectare a algoritmilor (reducere, divizare, greedy, programare dinamică)

Termen finalizare: 15.01.2022

Submiterea temelor: Se uploadează în Microsoft Teams o arhiva de tip *.zip având numele construit pe baza regulii:
NumeStudent_grupa_Tema1.zip (de exemplu `AdamEva_gr3_Tema2.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_grupa_Tema1_Problema (de exemplu `AdamEva_gr3_Tema2_Pb1.py`)

---

1. Se consideră trei tablouri de numere întregi, $a[1..n]$, $b[1..n]$, $c[1..n]$. Se pune problema verificării dacă există cel puţin un element comun în cele trei tablouri. De exemplu tablourile $a = [3, 1, 5, 10]$, $b = [4, 2, 6, 1]$, $c = [5, 3, 1, 7]$ au un element comun, pe când $a = [3, 1, 5, 10]$, $b = [4, 2, 6, 8]$, $c = [15, 6, 1, 7]$ nu au nici un element comun.
   (a) Propuneţi un algoritm de complexitate $O(n^3)$ care returnează `True` dacă cele trei tablouri conţin cel puţin un element comun şi `False` în caz contrar. Justificaţi faptul că algoritmul are complexitatea cerută şi implementaţi algoritmul în Python.
   (b) Propuneţi un algoritm de complexitate $O(n^2)$ care returnează `True` dacă cele trei tablouri conţin cel puţin un element comun şi `False` în caz contrar. Justificaţi faptul că algoritmul are complexitatea cerută şi implementaţi algoritmul în Python.
   (c) Presupunând că elementele tablourilor sunt din $\{1, 2, \ldots, m\}$ propuneţi un algoritm de complexitate $O(\max(m, n))$ care returnează `True` dacă cele trei tablouri conţin cel puţin un element comun şi `False` în caz contrar. Justificaţi faptul că algoritmul are complexitatea cerută şi implementaţi algoritmul în Python. *Indicaţie.* este permisă utilizarea unei zone suplimentare de memorie de dimensiune $\mathcal{O}(n)$.
   (d) Presupunând că toate cele trei tablouri sunt ordonate crescător propuneţi un algoritm de complexitate $O(n)$ care returnează `True` dacă cele trei tablouri conţin cel puţin un element comun şi `False` în caz contrar. Justificaţi faptul că algoritmul are complexitatea cerută şi implementaţi algoritmul în Python. *Indicaţie.* Se poate folosi ideea de la tehnica interclasării.

2. (a) Se consideră o matrice $A = (a_{ij})_{i=\overline{1,m}, j=\overline{1,n}}$ cu m linii şi n coloane. Propuneţi (şi implementaţi în Python) un algoritm care rearanjează liniile matricii astfel încât să fie în ordine crescătoare în raport cu norma euclidiană a liniilor (norma euclidiană a liniei i a matricii A este $\sqrt{a_{i1}^2 + a_{i2}^2 + \ldots + a_{in}^2}$). Stabiliţi ordinul de complexitate al algoritmului propus considerând ca operaţii dominante toate operaţiile efectuate asupra elementelor matricii (comparaţii, adunări, înmulţiri).
   (b) Se consideră o listă cu date de naştere specificate prin triplete de forma (zi, luna, an). Propuneţi (şi implementaţi în Python) un algoritm care ordonează crescător datele de naştere. Stabiliţi ordinul de complexitate al algoritmului propus.

3. Se pune problema implementării unor operaţii aritmetice cu numere reprezentate prin tablouri de n cifre (de exemplu, numărul 65189218901567 este reprezentat (pentru $n = 15$) prin $[0, 6, 5, 1, 8, 9, 2, 1, 8, 9, 0, 1, 5, 6, 7]$. Pentru fiecare dintre operaţiile următoare propuneţi un algoritm, stabiliţi ordinul de complexitate şi implementaţi-l în Python:

   (a) compararea a două numere a şi b reprezentate prin tablouri cu n elemente (algoritmul va returna $+1$ dacă a este mai mare decât b şi -1 în caz contrar).

   (b) calculul sumei a două numere a şi b reprezentate prin tablouri cu n elemente (rezultatul va fi reprezentat tot folosind un tablou cu n elemente, iar dacă valoarea obţinută depăşeşte zona alocată atunci se va returna un tablou cu toate elementele egale cu -1).

   (c) calculul diferenţei dintre două numere a şi b reprezentate prin tablouri cu n elemente (rezultatul va fi reprezentat tot folosind un tablou cu n elemente). Se presupune că $a \geq b$.

   (d) calculul produsului a două numere a şi b reprezentate prin tablouri cu n elemente (rezultatul va fi reprezentat tot folosind un tablou cu n elemente, iar dacă valoarea obţinută depăşeşte zona alocată atunci se va returna un tablou cu toate elementele egale cu -1).

4. 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.
   De exemplu matricea $[[3, 6, 10], [5, 9, 12], [11, 14, 16]]$ satisface proprietatea specificată. 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.

5. Se consideră doi algoritmi recursivi A şi B şi se presupune că timpii lor de execuţie satisfac următoarele relaţii de recurenţă: $T_A(n) = 7T_A(n/2) + n^2$ respectiv $T_B(n) = aT_B(n/4) + n^2$. Pentru ce valori ale lui a, algoritmul B are un ordin de complexitate mai mic decât algoritmul A?

   Indicaţie. Se poate folosi teorema Master pentru estimarea ordinului de complexitate al fiecărui algoritm.

6. Se consideră un set de n obiecte caracterizate prin dimensiunile $d_1 < d_2 < \ldots < d_n$ şi valorile $v_1 > v_2 > \ldots > v_n$ (ordonarea crescătoare după dimensiune corespunde cu ordonarea descrescătoare după valoare). Stiind că dimensiunile d_i au valori reale, propuneţi un algoritm care să selecteze un subset de obiecte care să încapă într-un rucsac de capacitate $C \in \mathbf{R}$ şi care să aibă valoarea maximă.

7. Se consideră o mulţime $A = \{a_1, a_2, \ldots, a_n\}$ cu elemente numere naturale, astfel încât $\sum_{i=1}^{n} a_i = 2k$. Să se determine o descompunere a lui A în două submulţimi disjuncte B şi C astfel încât $B \cup C = A$ şi suma elementelor din B, respectiv din C, să fie egală cu k.

   Indicaţie. Se reformulează problema ca una de optimizare cu restricţii pentru care poate fi aplicată tehnica programării dinamice: se caută o submulţime $S \subset A$ cu proprietatea că $\sum_{s \in A} s \leq k$ şi în acelaşi timp $\sum_{s \in A}$ este maximă. Problema este similară cu varianta discretă

a problemei rucsacului pentru care criteriul de optimizat este să rămână cât mai puţin spaţiu liber în rucsac (echivalent cu cazul în care valoarea obiectelor coincide cu dimensiunea lor).

8. Se consideră un set de activităţi A_1, A_2, ..., A_n, fiecare activitate, A_i, fiind caracterizată prin:

   (a) durata: $d_i \in \mathbf{N}$

   (b) termenul final de execuţie: $t_i \in \mathbf{N}$

   (c) profitul obţinut dacă se execută activitatea înainte de termenul final: $p_i \in \mathbf{R}$ (dacă activitatea nu este executată profitul este 0).

   Se pune problema determinării unei planificări a execuţiei activităţilor: $(T_1, T_2, \ldots, T_n)$, unde T_i reprezintă momentul startării activităţii A_i (dacă activitatea nu este executată atunci $T_i = -1$) astfel încât să fie îndeplinite condiţiile următoare:

   (a) activităţile planificate sunt compatibile între ele (nu pot fi executate două activităţi în acelaşi timp); două activităţi A_k şi A_l pot fi planificate dacă $T_k + d_k \leq \min\{t_k, T_l\}$ (în cazul în care $T_k < T_l$) respectiv $T_l + d_l \leq \min\{t_l, T_k\}$ (în cazul în care $T_l < T_k$);

   (b) profitul adus de activităţile planificate, $\sum_{k, T_k \neq -1} p_k$, este maxim.

   Propuneţi un algoritm bazat pe tehnica programării dinamice care permite determinare a uneia planificări optimale.

   Indicaţie. Presupunem că activităţile sunt ordonate crescător după termenul final de execuţie: $t_1 \leq t_2 \leq \ldots \leq t_n$ şi considerăm problema generică $P(i, j)$ a determinării unei planificări optimale pentru activităţile A_1, A_2, ..., A_i care se finalizează până la momentul j.
   (a) *Analiza unei soluţii optime şi identificarea subproblemelor.* Fie $(T_1, T_2, \ldots, T_i)$ o soluţie optimă a problemei $P(i, j)$. Sunt posibile două cazuri: (i) activitatea A_i nu este planificată ($T_i = -1$) ceea ce înseamnă că rezolvarea problemei $P(i, j)$ se reduce la rezolvarea problemei $P(i-1, j)$; (ii) activitatea A_i este planificată ($T_i \neq -1$) ceea ce înseamnă că rezolvarea problemei $P(i, j)$ se reduce la rezolvarea problemei $P(i-1, j-d_i)$;

   (b) *Construirea relaţiei de recurenţă.* Fie $R(i, j)$ profitul maxim obţinut prin planificarea activităţilor A_1, A_2, ..., A_i până la momentul j.

$$R(i,j) = \begin{cases} 0 & i = 0 \text{ sau } j = 0 \\ R(i-1,j) & \min\{j, t_i\} < d_i \\ \max_{0 \leq k \leq j-d_i}\{R(i-1,j), R(i-1,k) + p_i\} & \min\{j, t_i\} \geq d_i \end{cases}$$

   Pentru a facilita construirea soluţiei este util de reţinut pentru fiecare pereche (i, j) care este valoarea k pentru care se obţine valoarea maximă a profitului:

$$T(i,j) = \begin{cases} -1 & \text{dacă } R(i,j) = R(i-1,j) \\ kmax & \text{dacă } R(i-1,kmax) + p_i \geq R(i-1,k) + p_i, 0 \leq k \leq j - d_i \end{cases}$$

9. 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) Scrieţi relaţia de recurenţă care permite calculul scorului corespunzător unei alinieri optime.

(b) 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.

10. *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 $\{0, 1, \ldots, 2^{n-1}\}$.