Universitatea *Al. I. Cuza*, Iaşi `Numele:`
Facultatea de Informatică `Grupa:`

Proiectarea Algoritmilor - Test Scris (08 iunie 2018), seriile B + E

---

1. (12p) **Greedy**. Dându-se $n$ cursuri, fiecare cu un timp de început și un timp de final, să se găsească numărul minim de săli de curs necesare astfel încât să nu fie două cursuri simultan în aceeași sală.
   (a) (3p) Să se formuleze problema de mai sus ca pereche (*input*,*output*). Se vor da formulări cât mai precise şi riguroase.
   *Input*: $n \in \mathbb{N}$ – numărul de intervale; $s[0..n-1]$ – vector de numere naturale reprezentând pozițiile de început ale fiecărui interval; $f[0..n-1]$ – vector de numere naturale reprezentând pozițiile de sfârșit ale fiecărui interval cu proprietatea $s[i] \leq f[i]$ pentru orice $0 \leq i \leq n-1$.
   *Output*: $m \in \mathbb{N}$ - numărul de săli și $x[0..n-1]$ - vector de numere naturale între 0 și $m-1$ ($x[i]$ reprezentând indexul sălii în care a fost pus intervalul $i$) astfel încât $\forall i, j \in \{0, \ldots, n-1\}.x[i] = x[j] \rightarrow [s_i, f_i] \cap [s_j, f_j] = \emptyset$ și $m$ să fie minim.

   (b) (3p) Să se arate că strategia greedy care atribuie fiecărui curs, în ordinea crescătoare a timpilor de final, prima sală disponibilă nu conduce tot timpul la soluția optimă.
   Exemplu: pentru intervalele $[0, 10]$, $[5, 15]$, $[20, 30]$, $[12, 40]$, folosim 3 săli: $1, 2, 1, 3$. Soluția optimă folosește două săli: $1, 2, 2, 1$.

   (c) (4p) Să se descrie o strategie greedy care conduce la soluția optimă. Argumentați că strategia propusă produce soluția optimă.
   Se atribuie fiecărui curs, în ordinea crescătoare a timpilor de **început**, prima sală disponibilă.
   Argumentare: considerăm punctul $i$ din algoritm în care s-a folosit sala $m$ (cea mai mare). În acest punct, toate cursurile de la 0 la $i-1$ au primit o sală, iar cursul $i$ se suprapune cu $k-1$ cursuri. Deci există $k$ cursuri care se suprapun – și deci nu există soluție mai bună decât $k$.

   (d) (2p) Să se scrie în Alk un algoritm pentru problema de mai sus care implementează strategia greedy propusă la punctul anterior.

```
greedy(n, s, f) // presupun s in ordine crescatoare
{
  x = [];
  for (i = 0; i < n; ++i) {
    minim = n + 1; // nr mai mare decat orice sala
    for (j = 0; j < i; ++j) {
      if (f[j] >= s[i] && x[j] < minim) {
        minim = x[j];
      }
    }
    if (minim == n + 1) {
      x[i] = 0;
    } else {
      x[i] = minim + 1;
    }
  }
  return x;
}
```

---

*Ciornă.*

2. (12p) **Programare Dinamică**.
*Context*: Proiectarea unui algoritm, bazat pe paradigma programării dinamice, care găsește lungimea celei mai lungi 2-subsecvențe (subsecvență în care fiecare element este cel puțin dublul elementului de poziția anterioară). Exemplu: pentru șirul $2, 1, 3, 4, 8$, astfel de subsecvențe de lungime maximă sunt $2, 4, 8$ sau $1, 3, 8$ (și altele).
*Cerințe*:
   (a) (3p) Să se formuleze problema de mai sus ca pereche (*input*,*output*). Se vor da formulări cât mai precise şi riguroase.
   *Input*: $n$ – numărul de elemente ale șirului și $S[0..n-1]$ – șirul
   *Output*: cel mai mare număr $l$ a.î. $\exists i_0, \ldots, i_{l-1}$ cu proprietatea că $i_0 < i_1 < \ldots < i_{l-1}$ și $\forall j \in \{0, \ldots, l-2\}, S[i_j] \times 2 \leq S[i_{j+1}]$.

   (b) (3p) Fie $0 \leq i < n$ o poziție în șir, unde $n$ e lungimea șirului; notăm cu $d(i)$ lungimea celei mai lungi subsecvențe care satisface cerințele și care începe pe poziția $i$. Să se precizeze $d(n-1)$.

$$d(n-1) = 1.$$

   (c) (4p) Fie $0 \leq i < n-1$ o poziție în șir; șă se determine o relație de recurență pentru $d(i)$, în funcție de $d(i+1), d(i+2), \ldots, d(n-1)$.

$$d(i) = 1 + max(\{0\} \cup \{d(j) \mid i < j < n \wedge S[i] \times 2 \leq S[j]\})$$

   (d) (2p) Să se enunțe proprietatea de substructură optimă specifică problemei.
   Fie $i_0, \ldots, i_l$ o soluție optimă. Atunci $i_1, \ldots, i_l$ este o soluție optimă a subproblemei care începe pe poziția $i_1$.

---

*Ciornă.*

3. (12p) **Probleme NP-complete**.
Problema INDEPENDENT-SET.
*Input*: Un graf neorientat $G = (V, E)$, un număr natural $k$.
*Output*: "Da", dacă există $V' \subseteq V$ astfel încât $|V'| \geq k$ și, pentru orice două noduri $u, v \in V'$, muchia $u, v$ nu aparține mulțimii $E$.
"Nu", altfel.

(a) (3p) Să se definească clasa NP.
Clasa NP este clasa tuturor problemelor de decizie care pot fi rezolvate de un algoritm nedeterminist în timp polinomial în cazul cel mai nefavorabil.

(b) (3p) Să se arate că INDEPENDENT-SET $\in$ NP.
Este suficient să găsim un algoritm nedeterminist polinomial pentru INDEPENDENT-SET:

```
is(V, E, k)
{
  count = 0;
  for each v in V {
    choose x[v] in { 0 , 1 };
    count = count + x[v];
  }
  if (count < k) {
    failure;
  }
  for each u in V such that x[u] == 1 {
    for each v in V such that x[v] == 1 {
      if muchia {u,v} apartine E {
        failure;
      }
    }
  }
  success;
}
```

(c) (3p) Care dintre următoarele reduceri este suficientă pentru a arăta că problema INDEPENDENT-SET este NP-dificilă? De ce?
i. INDEPENDENT-SET la o problemă despre care știm deja că este NP-completă, în timp polinomial;
ii. o problemă despre care știm deja că e NP-completă la INDEPENDENT-SET, în timp polinomial.
Trebuie să găsim o reducere polinomială de la o problemă despre care știm deja că e NP-dificilă (e.g., CLIQUE) la INDEPENDENT-SET. CLIQUE fiind NP-dificilă, știm că orice problemă din NP se reduce polinomial la CLIQUE. Reducerea polinomială fiind tranzitivă, rezultă că orice problemă din NP se reduce polinomial la INDEPENDENT-SET.

(d) (3p) Să se arate că INDEPENDENT-SET este NP-completă (în rezolvare, puteți folosi fără demonstrație NP-completitudinea oricărei probleme NP-complete canonice, alta decât INDEPENDENT-SET, cum ar fi SAT, 3-SAT, CLIQUE, SUBSET SUM, VERTEX COVER, sau altele).
Știm de la punctul (2) că INDEPENDENT-SET $\in$ NP, deci este suficient să arătăm că este NP-dificilă. Arătăm că CLIQUE se reduce polinomial la INDEPENDENT-SET. Adică găsim un algoritm AlgClique polinomial pentru CLIQUE, presupunând că avem un algoritm AlgIS pentru INDEPENDENT-SET:

```
AlgClique(V,E,k)
{
  E' = emptyset; // calculam in E' complementul lui E
  for each u in V {
    for each v in V {
      if ((u != v) && ((not ({u, v} in E)))) {
        E'.insert({u, v});
      }
    }
  }
  return AlgIS(V,E',k);
}
```

---

*Ciornă.*

*Ciornă.*