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

Proiectarea Algoritmilor - Test Scris (06 iunie 2017), seriile A + E

---

1. (9p) **Greedy**. Problema Turului României. După sesiune, v-ați hotărât să faceți turul României, pornind din Iași. Aveți de vizitat n obiective pe parcurs, într-o ordine dată. Primul obiectiv se află la x_0 km distanță de Iași, al doilea la x_1 km după primul obiectiv, ș.a.m.d. (ultimul obiectiv, al n-lea, se află la distanță x_{n-1} km față de penultimul). Mașina poate parcurge k km cu un rezervor de benzină și puteți alimenta la oricare din cele n obiective (nu și pe drumul între obiective). Distanța dintre oricare două obiective consecutive este $\leq k$ (inclusiv distanţa la primul obiectiv). Când alimentați, faceți plinul de fiecare dată. Din Iași porniți cu rezervorul plin și vreți să minimizați numărul de alimentări de pe traseu.
   (a) (2p) Să se formuleze problema de mai sus ca pereche (*input*,*output*). Se vor da formulări cât mai precise şi riguroase.
   Input: $n, k \in \mathbb{N}$, $x[0..n-1]$ - vector de numere naturale reprezentând distanțele dintre obiective a.î. $\forall i \in \{0, \ldots, n-1\}, x[i] \leq k$.
   Output: cel mai mic număr l astfel încât $\exists i_1, \ldots, i_l \in \{0, \ldots, n-1\}$ și $x_0 + x_1 + \ldots + x_{i_1} \leq k$, $x_{i_1+1} + x_{i_1+2} + \ldots + x_{i_2} \leq k$, $\ldots$, $x_{i_l+1} + x_{i_l+2} + \ldots + x_{n-1} \leq k$.

   (b) (2p) Să se găsească un contraexemplu care arătă că strategia de a alimenta la fiecare obiectiv nu conduce la soluția optimă.
   Fie $n = 2, k = 10, x[0] = 1, x[1] = 2$. Cele două obiective pot fi vizitate cu un singur plin, deci strategia de a alimenta în toate cele 2 obiective nu este optimă.

   (c) (3p) Să se descrie o strategie greedy care conduce la soluția optimă. Argumentați că strategia propusă produce soluția optimă.
   La fiecare pas, se va alege alimentarea în obiectivul cel mai îndepărtat la care se ajunge cu benzina din rezervor.
   Argumentare: dacă există o soluție optimă astfel încât a i-a alimentare se face mai devreme, aceasta se poate transforma într-o soluție (tot optimă) în care a i-a alimentare se face conform strategiei greedy de mai sus.

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

```
greedy(n, k, x)
{
  result = 0;
  sum = 0;
  for (i = 0; i < n; ++i) {
    if (sum + x[i] > k) {
      sum = 0;
      result = result + 1;
    } else {
      sum = sum + x[i];
    }
  }
  return result;
}
```

Ciornă.

2. (9p) **Programare Dinamică**.
Context: proiectarea unui algoritm, bazat pe paradigma programării dinamice, care găsește lungimea celui mai lung palindrom care apare (nu neapărărat contiguu) într-un șir de caractere. De exemplu, pentru șirul FCS**IA**S**I**, răspunsul este 3 (palindromul poate fi, de exemplu, IAI, ISI, SAS, etc.).
Cerințe:

(a) (2p) Să se formuleze problema de mai sus ca pereche (*input*,*output*). Se vor da formulări cât mai precise şi riguroase.
Input: $n \in \mathbb{N}$, $S[0..n-1]$
Output: cel mai mare număr l a.î. $\exists i_1, \ldots, i_l$ cu proprietatea că $i_1 \leq i_2 \leq \ldots \leq i_l$ și $\forall j \in \{1, \ldots, l\}, S[i_j] = S[i_{l-j+1}]$.

(b) (2p) Fie i, j două poziții în șir a.î. $i \leq j$; notăm cu $d(i, j)$ lungimea celei mai lung palindrom (definit ca mai sus) care începe pe poziția i și se termină pe poziția j. Fie i o poziție. Să se precizeze $d(i, i)$ şi $d(i, i+1)$.

$$d(i,i) = 1$$

$$d(i, i+1) = \begin{cases} 2 & \text{dacă } S[i] = S[i+1] \\ 0 & \text{altfel} \end{cases} \quad \text{(pp. că și } i+1 \text{ este poziție)}e$$

(c) (3p) Fie i, j două poziții astfel încât $j > i + 1$. Să se determine o relație de recurență pentru $d(i, j)$, în funcție de $d(k, l)$, unde $i < k \leq l < j$.

$$d(i,j) = \begin{cases} 0 & \text{dacă } S[i] \neq S[j] \\ 2 + max_{i<k\leq l<j} d(k,l) & \text{altfel} \end{cases}$$

(d) (2p) Să se enunțe proprietatea de substructură optimă specifică problemei.
Fie i, j două poziții astfel încât $j > i+1$. Dacă soluția optimă pentru subproblema de a calcula $d(i, j)$ este $i = i_1, i_2, \ldots, i_u = j$ (de lungime u), atunci $i < i_2 \leq i_{u-1} < j$ și soluția optimă a subproblemei i_2, i_{u-1} este $i_2, \ldots, i_{u-1}$ (deci $d(i_2, i_{u-1}) = u - 2$).

Ciornă.

3. (9p) **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.
Problema CLIQUE.
Input: Un graf neorientat $G = (V, E)$, un număr natural k.
Output: "Da", dacă există $V' \subseteq V$ astfel încât $|V'| \geq k$ și există muchie între oricare două noduri din V'.
"Nu", altfel.

(a) (2p) 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 CLIQUE $\in$ NP.
Este suficient să găsim un algoritm nedeterminist polinomial pentru CLIQUE:

```
clique(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} nu apartine E {
        failure;
      }
    }
  }
  success;
}
```

(c) (2p) Care dintre următoarele reduceri este suficientă pentru a arăta că problema CLIQUE este NP-dificilă, știind că INDEPENDENT-SET este NP-dificilă? De ce?
i. CLIQUE la INDEPENDENT-SET, în timp polinomial;
ii. INDEPENDENT-SET la CLIQUE, în timp polinomial.
Trebuie să găsim o reducere polinomială de la INDEPENDENT-SET la CLIQUE. INDEPENDENT-SET fiind NP-dificilă, știm că orice problemă din NP se reduce polinomial la INDEPENDENT-SET. Reducerea polinomială fiind tranzitivă, rezultă că orice problemă din NP se reduce polinomial la CLIQUE.

(d) (2p) Să se arate că CLIQUE este NP-completă, știind că INDEPENDENT-SET este NP-dificilă.
Știm de la punctul (2) că CLIQUE $\in$ NP, deci este suficient să arătăm că este NP-dificilă. Arătăm că INDEPENDENT-SET se reduce polinomial la CLIQUE. Adică găsim un algoritm AlgIS polinomial pentru INDEPENDENT-SET, presupunând că avem un algoritm AlgClique pentru CLIQUE:

```
AlgIS(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 AlgClique(V,E',k);
}
```

Ciornă.

4. (9p) **Backtracking**.
Context: proiectarea unui algoritm de tip backtracking pentru problema `CLIQUE`. Presupunem că nodurile grafului sunt numerotate de la 0 la $n-1$. Vom reprezenta o soluție a problemei sub forma unui vector $v[0..n-1]$, unde:

$$v[i] = \begin{cases} 1 & \text{, dacă nodul } i \text{ aparține mulțimii } V' \\ 0 & \text{, altfel.} \end{cases}$$

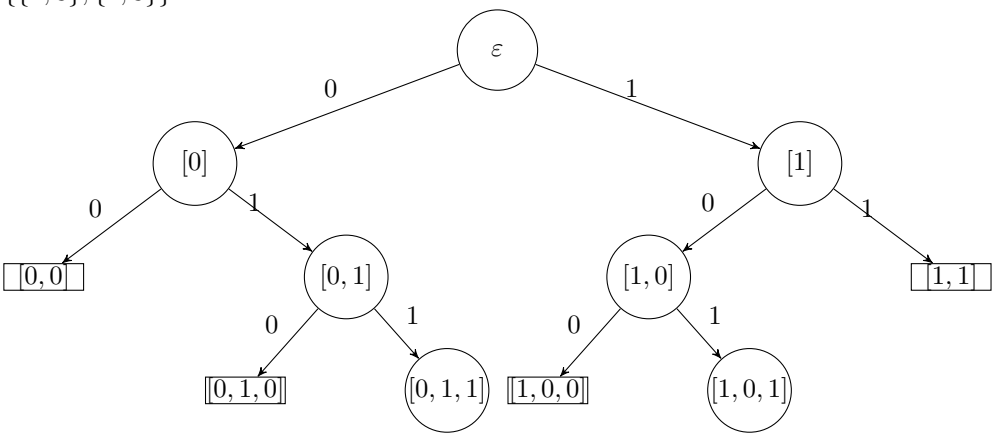
Cerințe:

(a) (3p) Să se definească noțiunea de *soluție parțială*.
O soluție parțială este dată de un prefix al unei soluții complete, adică de un vector $v[0..i-1]$, unde $0 \leq i \leq n$ și $v[j] \in \{0,1\}$ $(0 \leq j \leq i-1)$.

(b) (2p) Să se definească soluțiile parțiale *viabile*.
O soluție parțială $v[0..i-1]$ este viabilă dacă:
   i. pentru orice $0 \leq x, y \leq i-1$ a.î. $v[x] = v[y] = 1$, avem că există muchie între nodurile x și y;
   ii. numărul de 1 din vectorul $v[0..i-1]$ este $\geq k-(n-i)$ (trebuie să existe șansa de a selecta cel puțin k noduri).

(c) (3p) Să se definească *succesorii* unei soluții parțiale.
Dacă $i < n$, succesorii soluției parțiale $v[0..i-1]$ sunt vectorii $v_0[0..i]$ și $v_1[0..i]$ a.î. $v_0[0..i-1] = v_1[0..i-1] = v[0..i-1]$ și $v_0[i] = 0$, $v_1[i] = 1$.

(d) (1p) Să se reprezinte arborele de căutare $k = 2$ și pentru graful $G = (V, E)$, $V = \{1,2,3\}$, $E = \{\{1,3\},\{2,3\}\}$.



Ciornă.