Universitatea *Al. I. Cuza*, Iaşi
Facultatea de Informatică

`Numele:`
`Grupa:`

Proiectarea Algoritmilor - Test Scris (06 iunie 2017), seriile B + X

---

1. (9p) **Greedy**.
Problema Asfaltării. Pe o autostradă din România au apărut n gropi, la kilometrii $x_1, x_2, \ldots, x_n$. Compania de autostrăzi vrea să repare toate gropile, dar firmele de asfaltare sunt dispuse să repare sectoare de drum de exact k km. Dacă un sector începe la kilometrul x, atunci acesta cuprinde toate gropile aflate în intervalul $[x, x+k]$. Sectoarele pot să se suprapună.
Scopul este de a găsi o împărțire a gropilor în sectoare, astfel încât numărul de sectoare să fie minim.
   (a) (2p) Să se formuleze problema de mai sus ca pereche (*input*,*output*). Se vor da formulări cât mai precise şi riguroase.
   Input: $n \in \mathbb{N}$, $x[1..n]$ - pozițiile gropilor, ordonate crescător
   Output: cel mai mic număr natural l a.î. $\exists y_1, \ldots, y_l$ cu proprietatea că $\forall i \in \{1, \ldots, n\}. \exists j \in \{1, \ldots, l\}. x_i \in [y_j, y_j + k]$.

   (b) (2p) Să se găsească un contraexemplu care arată că strategia de a alege la fiecare pas un sector cu cât mai multe gropi nu conduce la soluția optimă.
   Pentru $k = 10$, $n = 7$, $x_1 = 0, x_2 = 8, x_3 = 9, x_4 = 10, x_5 = 11, x_6 = 12, x_7 = 20$, strategia va alege intervalul $[8, 18]$ (care conține 5 gropi) și va mai fi nevoie de încă două intervale pentru gropile $x_1 = 0$, $x_7 = 20$. În total, strategia propusă ar alege 3 intervale, în timp ce soluția optimă are doar 2: intervalele $[0, 10]$ și $[10, 20]$.

   (c) (3p) Să se descrie o strategie greedy care conduce la soluția optimă. Argumentați că strategia propusă produce soluția optimă.
   La fiecare pas, se alege un sector (de lungime k) care începe exact cu prima groapă neacoperită încă.
   Presupunând că ar exista o soluție optimă în care al i-lea sector (în ordinea crescătoare a punctelor de început) nu ar începe cu prima groapă neacoperită de primele $i-1$ sectoare, acesta ar putea fi putea "mutat" spre dreapta până la prima groapă neacoperită, fără a afecta optimalitatea soluției.

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

```
greedy(k, n, x)
{
  start = x[1];
  count = 1;
  for (i = 1; i <= n; ++i) {
    if (x[i] > start + k) {
      start = x[i];
      count++;
    }
  }
  return count;
}
```

Ciornă.

2. (9p) **Programare Dinamică**.
Context: proiectarea unui algoritm, bazat pe paradigma programării dinamice, care găsește lungimea celei mai mari subsecvențe contigue care apare atât în sens direct cât și invers într-un șir de caractere. De exemplu, pentru șirul R**EDI**V**XIDE**, cea mai lungă astfel de subsecvență are lungime 3 (secvența EDI, care apare și invers: IDE).
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. $\exists i, j$ cu proprietatea că $S[i..i+l-1] = S[j..j-l+1]$.

   (b) (2p) Fie i, j două poziții în șir; notăm cu $d(i, j)$ lungimea celei mai lungi subsecvențe contigue care apare atât în sens direct, pe pozițiile $i, i+1, \ldots, i+d(i,j)-1$, cât și în sens invers, pe pozițiile $j-d(i,j)+1, j-d(i,j)+2, \ldots, j$.
   Fie i o poziție. Să se determine valorile $d(n-1, i)$ și $d(i, 0)$.

$$d(n-1,i) = \begin{cases} 1 & \text{dacă } S[n-1] = S[i] \\ 0 & \text{altfel} \end{cases} \qquad d(i,0) = \begin{cases} 1 & \text{dacă } S[0] = S[i] \\ 0 & \text{altfel} \end{cases}$$

   (c) (3p) Fie $i < n-1$ și $j > 0$ două poziții în șir. Să se determine o relație de recurență pentru $d(i, j)$, în funcție de $d(i+1, j-1)$.

$$d(i,j) = \begin{cases} 0 & \text{dacă } S[i] \neq S[j] \\ 1 + d(i+1, j-1) & \text{dacă } S[i] = S[j] \end{cases}$$

   (d) (2p) Să se enunțe proprietatea de substructură optimă specifică problemei.
   Fie $i < n-1$, $j > 0$. Dacă $d(i, j) = l$ este lungimea cea mai mare a unei secvențe contigue care începe în i și se termină (în sens invers) în j și $l > 0$, atunci $d(i+1, j-1)$ este lungimea celei mai mari secvențe contigue care începe în $i+1$ și se termină (în sens invers) în $j-1$.

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 `VERTEX COVER`.
Input: Un graf neorientat $G = (V, E)$, un număr natural k.
Output: "Da", dacă există o submulțime $V' \subseteq V$ de noduri, de cardinal cel mult k, astfel încât V' să "acopere" toate muchiile din E: pentru orice muchie $\{u, v\}$ din E, măcar unul dintre nodurile u, v este în 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 `VERTEX COVER` $\in$ `NP`.
Este suficient să găsim un algoritm nedeterminist polinomial pentru `VERTEX COVER`:

```
vertexcover(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,v} in E {
    if ((x[u] == 0) && (x[v] == 0)) {
      failure;
    }
  }
  success;
}
```

(c) (2p) Care dintre următoarele reduceri este suficientă pentru a arăta că problema `VERTEX COVER` este `NP`-dificilă, știind că `INDEPENDENT-SET` este `NP`-dificilă? De ce?
  i. `VERTEX COVER` la `INDEPENDENT-SET`, în timp polinomial;
  ii. `INDEPENDENT-SET` la `VERTEX COVER`, în timp polinomial.

Trebuie să găsim o reducere polinomială de la `INDEPENDENT-SET` la `VERTEX-COVER`. `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 `VERTEX-COVER`.

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

```
AlgIS(V,E,k)
{
  return AlgVertexCover(V,E,|V|-k);
}
```

(Graful (V,E) are o acoperire cu noduri a muchiilor de dimensiune k ddacă are o mulțime independentă de dimensiune $n - k$, unde n este numărul de noduri).

Ciornă.

4. (9p) **Backtracking**.
Context: proiectarea unui algoritm de tip backtracking pentru problema `VERTEX COVER`. 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:

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

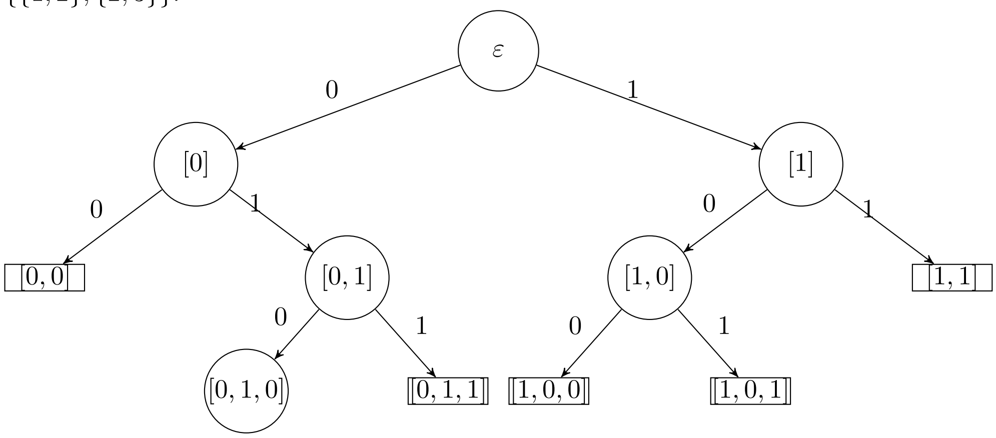
Cerințe:

(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 muchie x, y din E cu $x, y \in \{0, \ldots, i-1\}$, $v[x] = 1$ sau $v[y] = 1$;
   ii. numărul de noduri x a.î. $v[x] = 1$ este $\leq k$.

(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 pentru $k = 1$ și graful $G = (V, E)$, $V = \{1, 2, 3\}$, $E = \{\{1, 2\}, \{2, 3\}\}$.



Ciornă.