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

Proiectarea Algoritmilor - Test Scris (27 iunie 2017)

---

1. (18p) **Proiectare şi analiză, baza**.
Se consideră secvenţe s de numere întregi. Un element al unei secvenţe s este *k-majoritar* dacă apare de cel puţin $\left[\frac{n}{k}\right] + 1$ ori în s, unde n este lungimea secvenţei s şi $[x]$ partea întreagă a lui x. De exemplu, 5 este 3-majoritar în $(-1, 5, 2, 3, 5, -8, 5, 2)$, dar nu este şi 2-majoritar în aceeaşi secvenţă. Pentru o secvenţă s şi un număr natural k date, se pune problema determinării dacă s are un element k-majoritar, şi dacă da să se menţioneze un astfel de element. Notăm această problemă cu KMAJ.
   (a) (4p) Să se formuleze KMAJ ca pereche (*input*,*output*). Se vor da formulări cât mai precise şi riguroase.
   Input.
   $s = (s_0, \ldots, s_{n-1})$, $s_i \in \mathbb{Z}$ pentru $i = 0, \ldots, n-1$,
   $k \in \mathbb{N}, k \neq 0$;
   Output.
   x - dacă $|\{i \mid s_i = x, 0 \leq i < n\}| \geq \frac{n}{k} + 1$,
   none - dacă nu există x cu proprietatea $|\{i \mid s_i = x, 0 \leq i < n\}| \geq \frac{n}{k} + 1$.

   (b) (6p) Să se scrie un algoritm determinist care rezolvă KMAJ.
   Reprezentăm secvenţa s ca un tablou ($s_k = s[k]$).

```
kmaj(s, k) {
  n = s.size();
  for(i = 0; i < n; ++i) {
    nrap = 0;
    for (j = i; j < n; ++j)
      if (s[i] == s[j]) {
        ++nrap;
        if (nrap >= (n / k) + 1) return s[i];
      }
  }
  return "none";
}
```

   (c) (4p) Să se justifice că algoritmul rezolvă corect problema.
   Invariantul buclei `for` cu variabila contor `j`: `nrap` reprezintă numărul de apariţii ale lui $s[i]$ în subsecvenţa $s[i..j-1]$, adică `nrap` $= |\{k \mid s_k = s_i, i \leq k < j\}|$. Dacă `nrap` $\geq \left[\frac{n}{k}\right] + 1$ atunci s_i este k-majoritar.
   Invariantul buclei `for` cu variabila contor `j`: nu există x cu proprietatea $|\{k \mid s_k = x, 0 \leq k < i\}| \geq \frac{n}{k} + 1$.
   La terminarea lui `for` avem $i = n$ şi răspunsul "none" este returnat corect.

   (d) (4p) Să se calculeze complexitatea în cazul cel mai nefavorabil.
   Dimensiunea unei instanţe. n - numărul de elemente din secvenţa s;
   Operaţii numărate.
   comparaţiile `s[i] == s[j]`.
   Cazul cel mai nefavorabil.
   Când nu există element k-majoritar (algoritmul returnează "none").
   Timpul pentru cazul cel mai nefavorabil.
   Pentru un i fixat se execută $n - i$ comparaţii. Rezultă că numărul total de comparaţii în cazul cel mai nefavorabil este
   $T_{\texttt{kmaj}}(n) = n + (n-1) + \cdots 1 = \frac{n(n+1)}{2} = O(n^2)$.

Ciornă.

2. (18p) **Geometrie computaţională**.

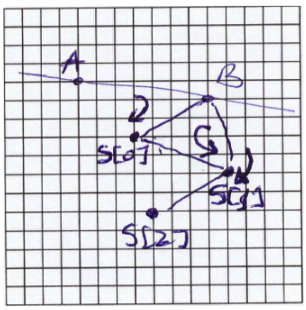
(a) (6p) Să se scrie în Alk un algoritm `las`(A, B, S), care

- are la intrare punctele din plan A, B (distincte) şi o mulţime de puncte S aflate de aceeaşi parte a dreptei determinate de A şi B şi
- determină la ieşire $C \in S$ cu proprietatea că unghiul $\widehat{ABC}$ este cel mai mare.

Reprezentăm S ca un tablou.

```
las(A, B, S) {
  C = S[0];
  k = ccw(A, B, C); // semiplanul in care se afla S
  for (i = 1; i < S.size(); ++i)
    if (ccw(B, C, S[i]) * k < 0) C = S[i];
  return C;
}
```

(b) (4p) Exemplificaţi execuţia algoritmului pe un exemplu.



i. $|S| = 3$;
ii. $k = ccw(A, B, S[0]) = -1$; $C = S[0]$;
iii. $i = 1$; $ccw(B, C, S[1]) = +1; \Rightarrow C = S[2]$;
iv. $i = 2$; $ccw(B, C, S[2]) = -1$;
Algoritmul returnează $C = S[1]$.

(c) (4p) Să se justifice corectitudinea algoritmului.
$\widehat{ABS[i]} > \widehat{ABC}$ dacă şi numai dacă
C se află în interiorul lui $\widehat{ABS[i]}$ dacă şi numai dacă
parcurgerile A, B, C şi $B, C, S[i]$ sunt contrare dacă şi numai dacă
$k \cdot ccw(B, C, S[i]) < 0$

(d) (4p) Să se indice cum se poate utiliza `las`(A, B, S) pentru determinarea înfăşurătorii convexe a unei mulţimi de puncte.
Algoritmul lui Jarvis:
Fie S mulţimea de puncte.
Se determina $L[0], L[1]$ din S astfel încât toate celelalte se găsesc de aceeaşi parte a dreptei $L[0]L[1]$;
$S = S \setminus \{L[0], L[1]\}$.
Pasul curent $i \geq 2$: $L[i] =$ `las`$(L[i-2], L[i-1], S)$; $S = \setminus \{L[i]\}$.
Criteriul de oprire: $L[i] == L[0]$.

Ciornă.

3. (18p) **Greedy**. Problema ordonării cursurilor. Aveți n cursuri numerotate de la 1 la n, iar al i-lea curs are $L[i]$ pagini. Dacă păstrați cursurile în ordine într-un singur teanc, *costul* de a accesa al k-lea curs este

$$\Sigma_{i=1}^{k} L[i],$$

deoarece trebuie să treceți prin toate foile primelor k cursuri pentru a ajunge la el. Știind că este la fel de probabil să accesați fiecare curs, costul mediu al accesării unui curs arbitrar este

$$\frac{1}{n} \times \Sigma_{k=1}^{n} \Sigma_{i=1}^{k} L[i].$$

Scopul este să rearanjați cursurile pe care le aveți astfel încât să *minimizați* costul mediu de a accesa un curs oarecare.

(a) (4p) Să se calculeze costul mediu de a accesa un curs dacă $n = 3, L[1] = 20, L[2] = 10, L[3] = 30$.
Costul este $(20 + (20 + 10) + (20 + 10 + 30))/3$.

(b) (4p) Să se găsească o reordonare a celor 3 cursuri de la punctul anterior care să minimizeze costul mediu de accesare al unui curs.
Optim ar fi să aranjăm cursurile în ordine $10, 20, 30$, obținând costul mediu $(10+(10+20)+(10+20+30))/3$.

(c) (4p) Să se formalizeze problema (cât mai precis) ca pereche input/output.
Input: $n \in \mathbb{N}$, $L[1..n] \in \mathbb{N}^n$
Output: o permutare $\pi : \{1, \ldots, n\} \to \{1, \ldots, n\}$ care să minimizeze $\frac{1}{n} \times \Sigma_{k=1}^{n} \Sigma_{i=1}^{k} L[\pi(i)]$.

(d) (6p) Să se descrie o strategie greedy de aranjare a cursurilor care să conducă la costul mediu de accesare al unui curs optim. Să se argumenteze pe scurt de ce strategie produce soluția optimă.
Este suficient să alegem la fiecare pas cursul cel mai scurt.
Procedând în acest fel, suma $L[1] + \ldots + L[\pi(i)]$ este mai mică sau egală cu $L[1] + \ldots + L[i]$ (pentru orice i), deci costul mediu este minimizat.

Ciornă.

4. (18p) **Probleme NP-complete și backtracking**.
Context: demonstrarea faptului că problema `INDEPENDENT-SET` este în clasa `NP` și proiectarea unui algoritm de tip backtracking pentru ea. Pentru algoritmul de backtracking, presupunem că nodurile grafului sunt numerotate de la 0 la $n-1$ ($V = \{0, 1, \ldots, n-1\}$) și vom reprezenta o soluție a problemei sub forma unui vector $v[0..n-1]$, unde:

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

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 (cu alte cuvinte, V' este o mulțime de cel puțin k noduri care nu conțin nicio muchie între ele).
"Nu", altfel.

(a) (6p) 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;
}
```

Algoritmul este nedeterminist, are complexitate $O(|V|^2|E|)$ și rezolvă problema `INDEPENDENT-SET`, deci această problemă, fiind de decizie, este în `NP`.

(b) (4p) Să se definească noțiunea de *soluție parțială* pentru problema `INDEPENDENT-SET`, ținând cont de definiția soluțiilor din *Context*.
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 $v[j] \in \{0, 1\}$ $(0 \leq j \leq i-1)$.

(c) (4p) Să se definească soluțiile parțiale *viabile* pentru problema `INDEPENDENT-SET`.
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 nu 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).

(d) (4p) Să se definească *succesorii* unei soluții parțiale pentru problema `INDEPENDENT-SET`.
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 $v_0[i] = 0$, $v_1[i] = 1$.

Ciornă.