Universitatea Al.I.Cuza Iasi **Nume:**
Facultatea de Informatică **Grupa:**

Proiectarea algoritmilor – Test scris 19.06.2014

Observaţii:

1. Nu este permisă consultarea bibliografiei.
2. Toate întrebările sunt obligatorii.
3. Fiecare întrebare/item este notată cu un număr de puncte indicat în paranteză.
4. Descrieţi conceptele utilizate în răspunsuri.
5. Algorimii vor fi descrişi în limbajul Alk. Se adimit extensii cu sintaxă inspirată din C++ (de exemplu, for, do-while, repeat-until, etc.). Pentru structurile de date utilizate (de exemplu, liste, mulţimi)se va preciza operaţiile (fără implementare daca nu se cere explicit) şi complexităţile timp şi spaţiu ale acestora.
6. Nu este permisă utilizarea de foi suplimentare.
7. Timp de răspuns: 1,25 ore.
8. **1.** În contextul algoritmului Knuth-Morris-Pratt (KMP) se consideră subiectul " bacbababababacab" şi patternul "ababaca".

a) [1] Să se calculeze funcția de eșec f pentru exemplul de mai sus. Explicați cum se calculează f.
b) [0.5] Să se reprezinte funcția eșec ca un automat.
c) [1] Să se explice cum se aplică algoritmul KMP pentru exemplul considerat. Câte comparații s-au efectuat?

Răspuns.

a)

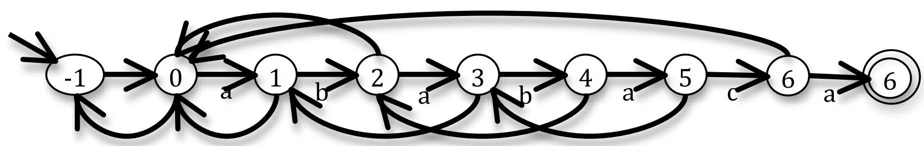
$f[j] = k$ dacă și numai dacă $p_0 \cdots p_{k-1}$ este de fapt cel mai lung prefix al lui p care se potrivește în p la poziția de sfârșit $j-1$. Funcția f poate fi definită recursiv:

...

```
determinaF (p, m, f) {
  f[0] = -1;
  for (j = 1;  j< m;  j = j+1)  {
    k = f[j-1];
    while ((k != -1) && (p[j-1] != p[k]))
      k = f[k];
    f[j] = k+1;
  }
}
```

	a	b	a	b	a	c	a
k	0	1	2	3	4	5	6
f[k]	-1	0	0	1	2	3	0

b)



c)

i pozitia in subiect, j pozitia in pattern:
Initial: i = j = 0
Pas 1: deoarece s[0] != p[0] si j != -1, j = f[0] = -1
Pas 2: deoarece j = -1, i = i+1 = 1, j = j+1 = 0
Pas 3: deoarece s[1] = p[0], i = i+1 =2, j = j+1=1
Pas 4: deoarece s[2] != p[1] si j != -1, j = f[j] = 0
Pas 5: deoarece s[2] != p[1] si j != -1, j = f[j] = -1
Pas 6: deoarece j = -1, i = i+1 = 3, j = j+1 = 0
...

```
KMP(s, n, p, m, f) {
  i = 0;
  j = 0;
  while (i < n) {
    while (j != -1) && (p[j] != s[i])
      j = f[j];
    if (j = m-1)
      return i-m+1; /* gasit p in s */
    else {
      i = i+1;
      j = j+1;
    }
  }
  return -1; /* p nu apare in s */
}
```

2. Se consideră următoarea problemă:
Intrare: un număr întreg x și o secvență de n numere întregi ordonate crescător $a_0 < a_1 < ... < a_{n-1}$.
Ieșire: i cu $a_i = x$ dacă există un astfel de i, -1 altfel.
a) [1] Să se scrie un algoritm care caută *secvențial* x în secvența a. Procesul de căutare se termină dacă se găsește i cu $a_i > x$.
b) [0.25] Să se definească dimensiunea unei instanțe pentru problema de mai sus.
c) [0.5] Să se scrie care este cazul cel mai nefavorabil.
d) [0.5] Să se determine complexitatea timp în cazul cel mai nefavorabil (se va considera măsura uniformă).
e) [1] Să se determine complexitatea timp medie (se va preciza cum se calculează complexitatea medie in general, care sunt valorile posibile pentru complexitate, ce distribuție de probabilitate se consideră, care este formula de calcul pentru algoritmul descris la a)).
f) [0.5] Ce se poate spune despre complexitatea problemei.
Răspuns.
a)
```
pos(a, n, x) {
  i = 0;
  while (i < n-1 && a[i] < x) i = i+1;
  if (a[i] == x) return i;
  return -1;
}
```
b) Definitia teoretica: g(p) = lumgimea instantei p = $\log x + \log a_0 + ... + \log a_{n-1} + \log n$
In cazul cand se considera masura uniforma, se ia g(p) = n = numarul de elemente din tablou.
c) cazul cel mai negavorabil este unul cand $x >= a_{n-1}$ deoarece in acest caz bucla while se executa de n-1 ori.
d)
initializare : 1 atribuire,
o bucla while: 2 comparatii + 1 adunare + 1 atribuire,
final 1 comapartie si returnarea.
In total: 1 + (n-1)*4 + 2 = 4*n + 7.
Avem 4 * n + 7 = O(n).
e)
- se considera $T_A(p)$ ca variabila aleatoare in care experienta este executia programului si rezultatul executiei este timpul de executie. $Tmed_A(n)$ este media varaibilei aleatoare $T_A(p)$ calculate pentru instantele p cu g(p)=n. Avem $Tmed_A(n) = \sum x_i q_i$, unde x_i sunt valorile posibile pentru $T_A(p)$, iar q_i probabilitatea cu care se obtine x_i.
- valorile posibile pentru $T_A(p)$: 4*i+3, i=0, ..., n-2 (cazul cand $a_i >= x$) si 4n+7 (cand $x >= a_{n-1}$)
- probabilitatea ca x sa apara in tablou = q; rezulta ca probab. ca x sa nu apara in a este 1-q iar probabilitatea sa apara pe poz i este q/n.
- $Tmed_A(n) = \sum(4*i+3)*q/n + (4*n+7)*(1-q)$
f) Algoritmul de mai sus arata ca problema are complexitatea O(n). Dar daca se considera un algoritm de cautare binara, atunci putem spune ca are complexitatea O(log n). In modelul arborilor de decizie pentru cautare are complexitaea Ω(log n).

Observatie: La a) se cerea algoritmul de cautare secventiala. Unii au scris algoritmul de cautare binara, care nu este o solutie corecta si deci nu s-a punctat.

2. În contextul algoritmilor de programare dinamică.
a) [0.5] Să se enunțe problema distanței (de editare) dintre șiruri (DES).
b) [0.5] Să se descrie noțiunea de *stare* pentru DES.
c) [0.75] Să se explice cum se aplică *principiul de optim* pentru a obține relația de recurență.
d) [1] Să se explice cum este utilizat algoritmul de programare dinamică pentru a calcula secvența de lungime minimă care transformă ”arca” în ”bara” .

Răspuns.

a)

> *Se consideră două șiruri $\alpha = a_1 \cdots a_n$ și $\beta = b_1 \cdots b_n$ formate cu litere dintr-un alfabet A. Asupra șirului α se pot face următoarele operații:*
> - Ștergere*: $S(i)$ – șterge litera de pe poziția i;*
> - Inserare*: $I(i, c)$ – inserează litera c pe poziția i;*
> - Modificare*: $M(i, c)$ – înlocuiește litera de pe poziția i cu c.*
>
> *Problema constă în determinarea unei secvențe de operații de lungime minimă care transformă pe α în β.*

b)

$DS(\alpha_i, \beta_j)$ corespunzătoare transformării subșirului $\alpha_i = a_1 \ldots a_i$ în $\beta_j = b_1 \ldots b_j$ și prin $d[i, j]$ valoarea optimă $d(\alpha_i, \beta_j)$.

c)

Considerăm decizia optimă prin care starea $DS(a_1 \ldots a_i, b_1 \ldots b_j)$ este transformată într-o stare $DS(a_1 \ldots a_{i'}, b_1 \ldots b_{j'})$ cu ($i' < i$ și $j' \leq j$) sau ($i' \leq i$ și $j' < j$).

Distingem următoarele situații:

1. Dacă $a_i = b_j$ atunci $i' = i - 1, j' = j - 1$ și, aplicând principiul de optim, obținem $d[i, j] = d[i - 1, j - 1]$.
2. $DS(a_1, \ldots, a_i, b_1, \ldots, b_j)$ se obține prin ștergere. Rezultă $i' = i - 1, j' = j$ și, aplicând principiul de optim, obținem $d[i, j] = d[i - 1, j] + 1$.
3. $DS(a_1, \ldots, a_i, b_1, \ldots, b_j)$ se obține prin inserare. Avem $i' = i, j' = j - 1$ și, aplicând principiul de optim, obținem $d[i, j] = d[i, j - 1] + 1$. Din corolarul lemei precedente rezultă că această operație poate fi realizată numai dacă $i < j$.
4. $DS(a_1, \ldots, a_i, b_1, \ldots, b_j)$ se obține prin modificare. Avem $i' = i - 1, j' = j - 1$ și, aplicând principiul de optim, obținem $d[i, j] = d[i - 1, j - 1] + 1$.

$$d[i, j] = \min\{d[i - 1, j] + 1, d[i - 1, j - 1] + \delta, d[i, j - 1] + 1\}$$

unde

$$\delta = \begin{cases} 0 & , \text{dacă } a_i = b_j \\ 1 & , \text{dacă } a_i \neq b_j \end{cases}$$

d)
Initial: p[0,j] =j, p[i,0] =i, pentru i,j=0,1,2,3,4
Pas 1: d[1,1] = d[0,0] + 1 = 1 deoarece alfa[1] != beta[1]
Pas 2: d[1,2] = d[1,1] + 1 = 2.
...

Observatie: daca se da matricea d, atunci trebuie explicat (macar pe un exemplu) cum sunt calculate aceste valori.