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

Proiectarea algoritmilor – Test scris 11.04.2014, A

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. Descrieți conceptele utilizate în răspunsuri.
4. Algorimii vor fi descrişi în limbajul Alk. Se admit extensii cu sintaxă inspirată din C++ (de exemplu, for, do-while, repeat-until, etc.). Pentru structurile de date utilizate (de exemplu, liste, mulțimi)se va preciza operațiile (fără implementare daca nu se cere explicit) și complexitățile timp și spațiu ale acestora.
5. Nu este permisă utilizarea de foi suplimentare.
6. Timp de răspuns: 1 oră.

1. În contextul algoritmului Boyer-Moore se consideră subiectul "E CARATA DAR NU E ARATATA" și patternul "ARATAT".
a) [0.5] Să se enunțe regula caracterului rău (bad character rule).
b) [0.5] Presupunând ca alfabetul e format din literele care apar în cele două șiruri, să se calculeze funcția de salt.
c) [0.5] Să se enunțe regula celui mai bun sufix.
d) [0.5] Să se calculeze valorile goodSuff (j).
e) [1] Să se explice cum se aplică algoritmul Boyer-Moore pentru exemplul considerat. Câte comparații s-au efectuat?

Răspuns.

a)

Dacă $p[j] \neq s[i] = C$,

1. dacă apariția cea mai din dreapta a lui C în p este $k < j$, $p[k]$ și $s[i]$ sunt aliniate
2. dacă apariția cea mai din dreapta a lui C în p este $k > j$, p este translatat la dreapta cu o poziție
3. dacă C nu apare în p, patternul p este aliniat cu $s[i+1..i+m]$ (saltul $= j$)

b)

$$salt[C] = \begin{cases} m - \text{poziţia ultimei apariţii a lui } C \text{ în pattern} & \text{, dacă } C \text{ apare în pattern} \\ m & \text{, altfel} \end{cases}$$

(alternativ, $salt(C) = \max(\{0\} \cup \{i < m \mid p[i] = C\})$)

	A	C	D	E	N	R	T	U
6	1	6	6	6	6	4	2	6

c)

Presupunem că $p[j-1]$ nu se potrivește (după ce s-au potrivit $p[j..m-1]$.

1. dacă $goodSuff(j) > 0$, face un salt egal cu $m - goodSuff(j)$ (cazul 1)
2. dacă $goodSuff(j) = 0$, face un salt egal cu $m - lp(j)$ (cazul 1)

Dacă $p[m-1]$ nu se potrivește, atunci $j = m$ și saltul este corect.

unde

$lp(j)$ = lungimea celui mai lung prefix al lui p care este sufix al lui $p[j..m-1]$.

d)

$goodSuff(j)$ = poziția de sfârșit a unei apariții a lui $p[j..m-1]$ care nu este precedată de $p[j-1]$

A	R	A	T	A	T
0	1	2	3	4	5
0	0	0	0	3	0

e)

daca exista nepotrivire pe pozitia p[j], mareste k (= pozitia de inceput al lui i la urmatoarea iteratie) cu maximul dintre salturile date de regula caracterului rau si regula sufixului bun.

E		C	A	R	A	...
A	R	A	T	A	T	

se aplica regula caracterului rau, se incrementeaza pozitia i din text cu salt[A]=1 (se aliniaza A) [1 comparatie]

	C	A	R	A	T	A	...
A	R	A	T	A	T		

se aplica regula caracterului rau, se incrementeaza pozitia i cu salt[R]=4 (se aliniaza R) (3 comparatii)

...

2. În contextul algoritmilor *greedy*.
a) [0.5] Să se enunțe problema codurilor Huffman.
b) [0.5] Să se explice legătura dintre codurile Huffman și arborii binari.
c) [0.5] Să se explice cine sunt mulțimea de stări S și colecția de submulțimi C.
d) [0.5] Să se descrie pasul de alegere locală dinalgoritmul greedy care construiește arborele Huffman.
e) [1] Să se explice cum este utilizat algoritmul greedy pentru a construi o codificare Huffman pentru textul ”streets are never stars” (mesajele sunt carcactere care apar în text, frecvența este dată de numărul de apariții).
Răspuns.
a)
Intrare

- n mesaje $M_0, M_1, ..., M_{n-1}$ cu frecventele $w_0, w_1, ..., w_{n-1}$ codificate astfel incat $cod(M_i) \in \{0,1\}^*$, $\forall i,j: i \neq j \Rightarrow cod(M_i)$ nu este prefix a lui $cod(M_j)$; lungimea medie a codificarii = $1/n \sum_{i=0,n-1} (|cod(M_i)|w_i)$

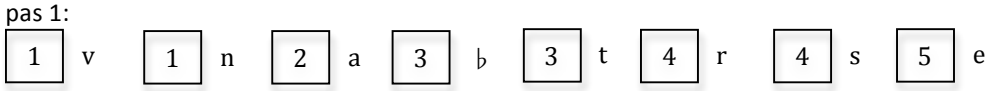
Iesire

- codificare cu lungimea medie minima

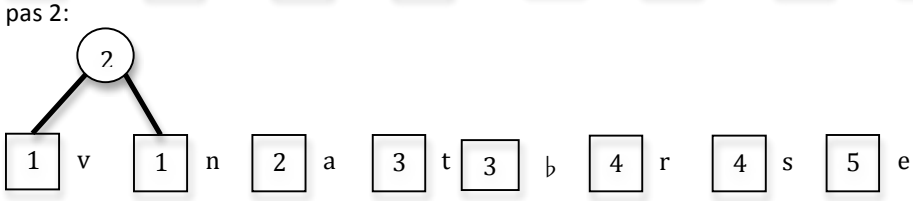
b)
Codurile pot fi memorate de un arbore binar a.i. orice cod descrie unic un drum de la radacina la frontiera.
Regula de parcurgere: cod = 0, se coboara la copilul stang; cod = 1, se coboara la copilul drept.
Un exemplu.
c)
S – cea mai mica multime de arbori construita astfel:
1) $w_i \in S$ pentru orice i
2) $T_1, T_2 \in S \Rightarrow T_1 \oplus T_2 \in S$
De explicat operatia $\oplus$
$X \in C$ daca:
1) $(\forall T_1, T_2 \in X) \neg (T_1 \leq T_2 \,||\, T_2 \leq T_1)$, unde $T_1 \leq T_2$ ddaca exista T a. i. $T_2 = T_1 \oplus T$
2) X este finita
d)
alege T_1, T_2 cu radacini minime in B si $T_1 \oplus T_2$ nu este in B
$B = B - \{T_1, T_2\} \cup \{T_1 \oplus T_2\}$
e)
- se calculeaza mai intai frecventele (␣ este caracterul spatiu)

a	e	n	r	s	t	v	␣
2	5	1	4	4	3	1	3

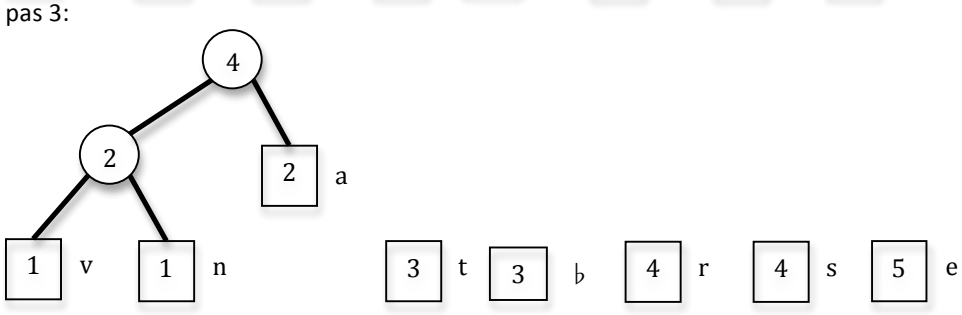
pas 1:



pas 2:



pas 3:



...

3. Se consideră problema Submulțime de sumă dată (SSD):
Intrare: o mulțime A astfel orice $a \in A$ are o mărime $s[a] \in Z_+$, și un număr $M \in Z_+$.
Ieșire: cel mai mare $M^* \leq M$ a.î. există $A' \subseteq A$ cu $\sum(s[a] \mid a \in A') = M^*$.
a) [1] Să se arate că SSD este în NP. Justificare. Indicatie (pe tabla): se va arata ca varianta ca problema de decizie este in NP.
b) [0.5] Există algoritm determinist polinomial care rezolvă SSD? Justificare.
c) [1] Să se dea un exemplu de algoritm care calculează o aproximare a soluției optime. Ce se poate spune despre aproximare?
d) [0.5] Precizați complexitatea timp a algoritmului de aproximare.
Răspuns.
a)
- se considera varianta data ca problema decizie, care cere daca se poate alege A' a.i. M* = M si notata SSDDec
- trebuie dat un algoritm nedeterminist care rezolva SSDDec

```
nondetSSDDec (A, s, M) {
  foreach (a in A) x[a] = choice(2); // partea de ghicire
  //partea de verificare
  mStar = 0;
  foreach (a in A) mStar = mStar + x[a];
  if (mStar = M) return succes;
  else return failure;
}
```

b) Oricare din cele doua variante este problema NP-completa. Deoarece nu se stie daca P = NP, nu se poate da un raspuns exact. Dar se stie ca pana acum nu s-a gasit pentru niciuna din probleme algoritm determinist polinomial care sa o rezolve.
c)

```
ssdAprox(n, s, M, ε) {
  L[0] = (0);
  for (i = 1; i <= n; ++i) {
    Ltemp = merge(L[i-1], L[i-1] + s[i]);
    curata(Ltemp, ε/n, L[i]);
    elimina din L[i] valorile mai mari decit M;
  }
  return cea mai mare valoare din L[n];
}
```

unde curata(Ltemp, ε/n, L[i]) pune in L[i] lista Ltemp curatata de delta = ε/n.
- este o schema de aproximare polinomial completa, adica eroarea relativa este marginita de ε si complexitatea timp este polinomiala ata in n cat si 1/ε.
d)
Complexitatea timp va fi $O(n \max_i L[i].length())$. Se stie ca $L[i].length() = O(n \ln M / \varepsilon)$ (teorema de la curs).