Proiectarea algoritmilor
Tema de casă nr. 2
Paradigma *greedy*
Compresii de date

Cuprins

1 Arbori binari ponderaţi pe frontieră

1.1 Descriere

Considerăm arbori binari cu proprietatea că orice vârf are 0 sau 2 succesori şi vârfurile de pe frontieră au ca informaţii (etichete, ponderi) numere, notate cu $info(v)$. Convenim să numim aceşti arbori ca fiind *ponderaţi pe frontieră*. Pentru un vârf v din arborele t notăm cu d_v lungimea drumului de la rădăcina lui t la vârful v. *Lungimea externă ponderată* a arborelui t este:

$$\mathrm{LEP}(t) = \sum_{v \text{ pe frontiera lui } t} d_v \cdot info(v)$$

Modificăm aceşti arbori etichetând vârfurile interne cu numere ce reprezintă suma etichetelor din cele două vârfuri fii. Pentru orice vârf intern v avem $info(v) = info(v_1) + info(v_2)$, unde v_1, v_2 sunt fiii lui v (Figura 1).

Lema 1.1 *Fie t un arbore binar ponderat pe frontieră. Atunci:*

$$\mathrm{LEP}(t) = \sum_{v \text{ intern în } t} info(v)$$

Lema 1.2 *Fie t un arbore din $\mathscr{T}(x)$ cu LEP minimă şi v_1, v_2 două vârfuri pe frontiera lui t. Dacă $info(v_1) < info(v_2)$ atunci $d_{v_1} \geq d_{v_2}$.*

Lema 1.3 *Presupunem $x_0 \leq x_1 \leq \cdots \leq x_{n-1}$. Există un arbore în $\mathscr{T}(x)$ cu LEP minimă şi în care vârfurile etichetate cu x_0 şi x_1 (vârfurile sunt situate pe frontieră) sunt fraţi.*

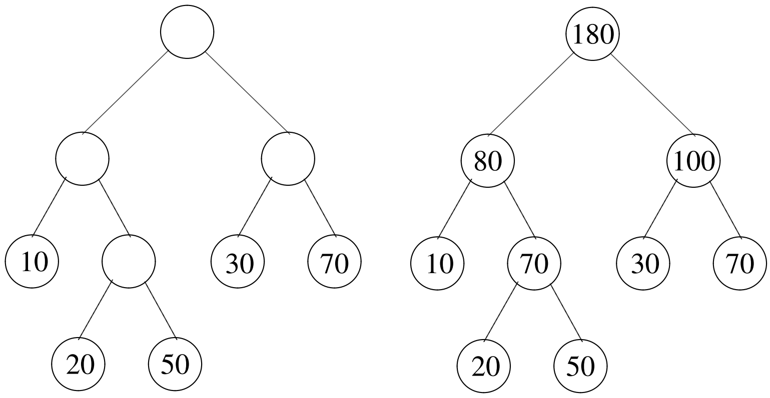


Figura 1: Arbore ponderat pe frontieră, înainte şi după modificare

1.2 Algoritm pentru construirea unui arbore cu lungimea externă ponderată minimă

Ideea algoritmului rezultă direct din Lema 1.3. Presupunem $x_0 \leq x_1 \leq \cdots \leq x_{n-1}$. Ştim că există un arbore optim t în care x_0 şi x_1 sunt memorate în vârfuri frate. Tatăl celor două vârfuri va memora $x_0 + x_1$. Prin ştergerea celor două vârfuri ce memorează x_0 şi x_1 se obţine un arbore t'. Fie $t1'$ un arbore optim pentru secvenţa $y = (x_0 + x_1, x_2, \ldots, x_{n-1})$ şi $t1$ arborele obţinut din $t1'$ prin "agăţarea" a două vârfuri cu informaţiile x_0 şi x_1 de vârful ce memorează $x_0 + x_1$. Avem $\text{LEP}(t1') \leq \text{LEP}(t')$ ce implică $\text{LEP}(t1) = \text{LEP}(t1') + x_0 + x_1 \leq \text{LEP}(t') + x_0 + x_1 = \text{LEP}(t)$. Cum t este optim, rezultă $\text{LEP}(t1) = \text{LEP}(t)$ şi de aici t' este optim pentru secvenţa y. Considerăm în loc de secvenţe de numere secvenţe de arbori.

Notaţii: $t(x_i)$ = arborele format dintr-un singur vârf etichetat cu x_i si $rad(t)$ = rădăcina arborelui t.

Premise: Iniţial se consideră n arbori cu un singur vârf, care memorează numerele $x_i, i = 0, \ldots, n-1$.

```
procedure lep(x, n)
   1:  B ← {t(x_0),...,t(x_{n-1})}
   2:  while (#B > 1) do
   3:      alege t1, t2 din B cu info(rad(t1)),info(rad(t2)) minime
   4:      construieşte arborele t în care subarborii rădăcinii
   5:        sunt t1, t2 şi info(rad(t))=info(rad(t1))+info(rad(t2))
   6:      B ← (B \ {t1,t2}) ∪ {t}
end
```

1.3 Implementarea algoritmului pentru construirea unui arbore cu lungimea externă ponderată minimă

a) Dacă mulţimea B este implementată printr-o listă liniară, atunci în cazul cel mai nefavorabil operaţia 3 este are timpul de execuţie $O(n)$, iar operaţia 6 are timpul de execuţie $O(1)$.

b) Dacă mulţimea B este implementată printr-o listă liniară ordonată, atunci în cazul cel mai nefavorabil operaţia 3 are timpul de execuţie $O(1)$, iar operaţia 6 are timpul de execuţie $O(n)$.

c) Dacă mulţimea B este implementată printr-un *heap*, atunci în cazul cel mai nefavorabil operaţia 3 are timpul de execuţie $O(\log n)$, iar operaţia 6 are timpul de execuţie $O(\log n)$.

Concluzie: *heapul* este alegerea cea mai bună pentru implementarea mulţimii B.

2 Compresii de date

2.1 Descriere

Fie n mesaje $M_0, \ldots, M_{n-1}$ recepţionate cu frecvenţele $f_0, \ldots, f_{n-1}$. Mesajele sunt codificate cu şiruri (cuvinte) construite peste alfabetul $\{0, 1\}$ cu proprietatea că pentru orice $i \neq j$, codul mesajului M_i nu este un prefix al codului lui M_j. O astfel de codificare se numeşte *independentă de prefix* („prefix-free”).

Notăm cu d_i lungimea codului mesajului M_i. *Lungimea medie* a codului este $\sum_{i=0}^{n-1} f_i \cdot d_i$. Problema constă în determinarea unei codificări cu lungimea medie minimă.

Unei codificări îi putem asocia un arbore binar cu proprietăţile următoare:

- Mesajele corespund nodurilor de pe frontieră.
- Muchiile (*tata, fiu-stânga*) sunt etichetate cu 0;
- Muchiile (*tata, fiu-dreapta*) sunt etichetate cu 1.
- Nodurile de pe frontiera arborelui sunt etichetate cu frecvenţele mesajelor corespunzătoare.

Drumul de la rădăcină la un nod de pe frontieră descrie codul mesajului asociat acestui nod.

Determinarea unui cod optim coincide cu determinarea unui arbore ponderat pe frontieră optim.

2.2 Coduri Huffman

Codurile Huffman pot fi utilizate la scrierea comprimată a textelor. Considerăm textul HARABABURA. Mesajele sunt literele din text, iar frecvenţele sunt date de numărul de apariţii ale fiecărei litere în text (Figura 2a).

Literă	Frecvenţă
H	1
A	4
R	2
B	3
U	1

a)

Literă	Cod
H	010
A	1
R	000
B	001
U	011

b)

Figura 2: Codificarea caracterelor din textul HARABABURA

Presupunem că intrarea este memorată într-un tablou `T` de structuri cu două câmpuri:

1. `T[i].mes` conţine mesajul M_i;
2. `T[i].f` conţine frecvenţa f_i.

Algoritmul care urmează utilizează reprezentarea arborilor prin tablouri de structuri.

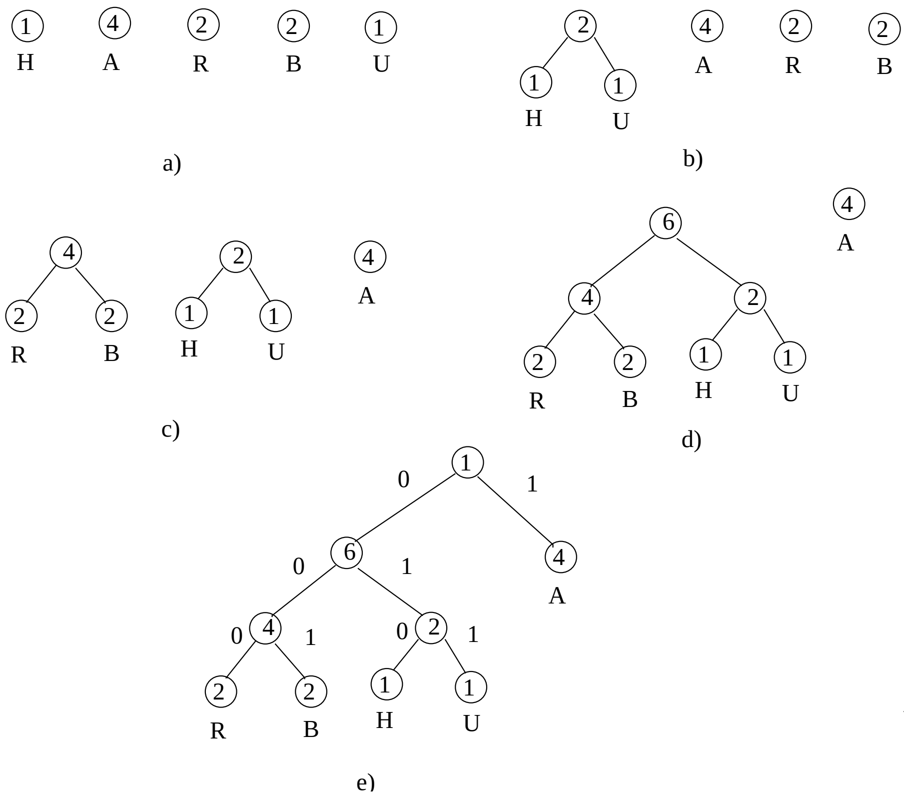
Notăm cu `H` tabloul ce reprezintă arborele Huffman. Tabloul `H` conţine structuri formate din trei câmpuri:

1. `H[i].elt`;
2. `H[i].fst` = indicele fiului de pe partea stângă;
3. `H[i].fdr` = indicele fiului de pe partea dreaptă.

Semnificaţia câmpului H[i].elt este următoarea:

1. dacă i este nod intern, atunci H[i].elt reprezintă informaţia calculată din nod;

2. dacă i este pe frontieră (corespunde unui mesaj), atunci H[i].elt este adresa din T a mesajului corespunzător.



Literă	Cod
H	010
A	1
R	000
B	001
U	011

Figura 3: Construcţia arborelui Huffman pentru HARABABURA

Notăm cu val(i) funcţia care intoarce informaţia din nodul i, calculată ca mai sus.

Tabloul H, care în final va memora arborele de interclasare optimală, va memora pe parcursul construcţiei acestuia colecţiile intermediare de arbori.

În timpul execuţiei algoritmului de construcţie a arborelui, H este compus din trei părţi (Figura 4):

Partea I: un min-*heap* care va conţine rădăcinile arborilor din colecţie;

Partea a II-a: conţine nodurile care nu sunt rădăcini;

Partea a III-a: zonă vidă în care se poate extinde partea din mijloc.

heap-ul rădăcinilor	noduri care nu nu sunt rădăcini	zonă vidă

Figura 4: Organizarea tabloului H

Un pas al algoritmului de construcţie ce realizează selecţia *greedy* presupune parcurgerea următoarelor etape:

1. Mutarea rădăcinii cu informaţia cea mai mică pe prima poziţie liberă din zona a treia, să zicem k. Aceasta este realizată de următoarele operaţii:

   a) copierea rădăcinii de pe prima poziţie din heap pe poziţia k:

   ```
   H[k] ← H[1]
   k ← k + 1
   ```

   b) mutarea ultimului element din heap pe prima poziţie:

   ```
   H[1] ← H[m]
   m ← m - 1
   ```

   c) refacerea min-*heapului*.

2. Copierea rădăcinii cu informaţia cea mai mică pe prima poziţie liberă din zona a treia, fără a o elimina din min-*heap*:

   ```
   H[k] ← H[1]
   k ← k + 1
   ```

3. Construirea noii rădăcini şi memorarea acesteia pe prima poziţie în min-*heap* (în locul celei copiate anterior).

4. Refacerea min-*heapului*.

Algoritmul rezultat are timpul de execuţie $O(n \log n)$.

3 Sarcini de lucru şi barem de notare

Sarcini de lucru:

1. Dat fiind un text memorat într-un fişier, scrieţi un program C/C++ care codifică binar textul astfel încât lungimea medie a codului rezultat sĕfie minimă (Vezi algoritmul `lep`, figura 4 şi algoritmul de construcţie a arborelui Huffman optim).

Barem de notare:

1. Implementarea algoritmului algoritmul de construcţie a arborelui Huffman optim: 6p
2. Codificarea binară a textului astfel încât lungimea medie a codului rezultat să fie minimă: 3p
3. Baza: 1p

Bibliografie

[1] Lucanu, D. şi Craus, M., *Proiectarea algoritmilor*, Editura Polirom, 2008.

[2] Moret, B.M.E.şi Shapiro, H.D. , *Algorithms from P to NP: Design and Efficiency*, The Benjamin/Cummings Publishing Company, Inc., 1991.