

Proiectarea algoritmilor

Tema de casă nr. 2

Cuprins

Arbori binari ponderați pe frontieră

- Descriere

- Algoritm pentru construirea unui arbore cu lungimea externă ponderată minimă

Compresii de date

- Descriere

- Coduri Huffman

Sarcini de lucru și barem de notare

Bibliografie

Arbori binari ponderați pe frontieră - descriere

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

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

- Modificăm acești 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).

Arbori binari ponderați pe frontieră - exemplu

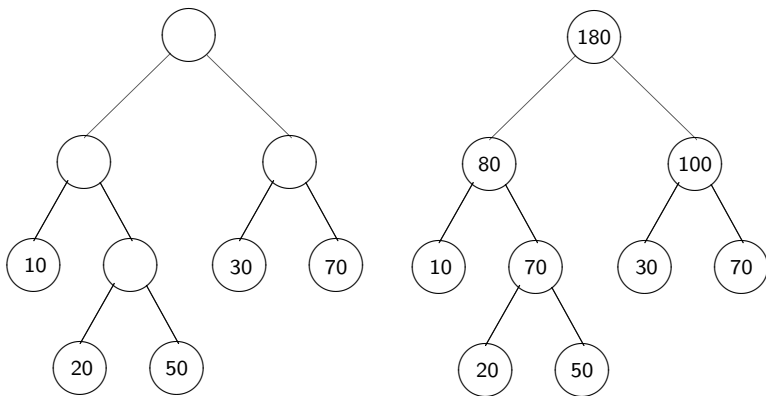


Figura 1 : Arbore ponderat pe frontieră, înainte și după modificare



Lungimea externă ponderată

Lemma (1)

Fie t un arbore binar ponderat pe frontieră.

Atunci

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

Lemma (2)

Fie t un arbore din $\mathcal{T}(x)$ cu LEP minimă și v_1, v_2 două vârfuri pe frontiera lui t . Dacă $\text{info}(v_1) < \text{info}(v_2)$ atunci $d_{v_1} \geq d_{v_2}$.

Lemma (3)

Presupunem $x_0 \leq x_1 \leq \dots \leq x_{n-1}$. Există un arbore în $\mathcal{T}(x)$ cu LEP minimă și în care vârfurile etichetate cu x_0 și x_1 (vârfurile sunt situate pe frontieră) sunt frați.

Algoritm pentru construirea unui arbore cu lungimea externă ponderată minimă - descriere

- Ideea algoritmului rezultă direct din Lema 3.
- Presupunem $x_0 \leq x_1 \leq \dots \leq x_{n-1}$.
- Știm că există un arbore optim t în care x_0 și x_1 sunt memorate în vârfuri frate. Tatăl celor două vârfuri va memora $x_0 + x_1$.
- Prin ștergerea celor două vârfuri ce memorează x_0 și x_1 se obține un arbore t' .
- Fie $t1'$ un arbore optim pentru secvența $y = (x_0 + x_1, x_2, \dots, x_{n-1})$ și $t1$ arborele obținut din $t1'$ prin „agățarea” a două vârfuri cu informațiile x_0 și 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 $\text{LEP}(t1) = \text{LEP}(t)$ și de aici t' este optim pentru secvența y .



Algoritm pentru construirea unui arbore cu lungimea externă ponderată minimă - pseudocod

- Considerăm în loc de secvențe de numere secvențe de arbori.
- *Notatii:* $t(x_i)$ desemnează arborele format dintr-un singur vârf etichetat cu x_i iar $rad(t)$ rădăcina arborelui t .
- *Premise:* Inițial se consideră n arbori cu un singur vârf, care memorează numerele $x_i, i = 0, \dots, n-1$.

procedure lep(x, n)

1: $B \leftarrow \{t(x_0), \dots, t(x_{n-1})\}$

2: while (#B > 1) do

3: alege t_1, t_2 din B cu $info(rad(t_1)), info(rad(t_2))$ minime

4: construiește arborele t în care subarborii rădăcinii

5: sunt t_1, t_2 și $info(rad(t)) = info(rad(t_1)) + info(rad(t_2))$

6: $B \leftarrow (B \setminus \{t_1, t_2\}) \cup \{t\}$

end



Implementarea algoritmului pentru construirea unui arbore cu lungimea externă ponderată minimă

- a) Dacă mulțimea B este implementată printr-o listă liniară, atunci în cazul cel mai nefavorabil operația 3 are timpul de execuție $O(n)$, iar operația 6 are timpul de execuție $O(1)$.
- b) Dacă mulțimea B este implementată printr-o listă liniară ordonată, atunci în cazul cel mai nefavorabil operația 3 are timpul de execuție $O(1)$, iar operația 6 are timpul de execuție $O(n)$.
- c) Dacă mulțimea B este implementată printr-un *heap*, atunci în cazul cel mai nefavorabil operația 3 are timpul de execuție $O(\log n)$, iar operația 6 are timpul de execuție $O(\log n)$.

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

Codificare de lungimea medie minimă.

- Fie n mesaje M_0, \dots, M_{n-1} recepționate cu frecvențele f_0, \dots, f_{n-1} .
- Mesajele sunt codificate cu șiruri (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 numește *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 proprietățile 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 frecvențele 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.

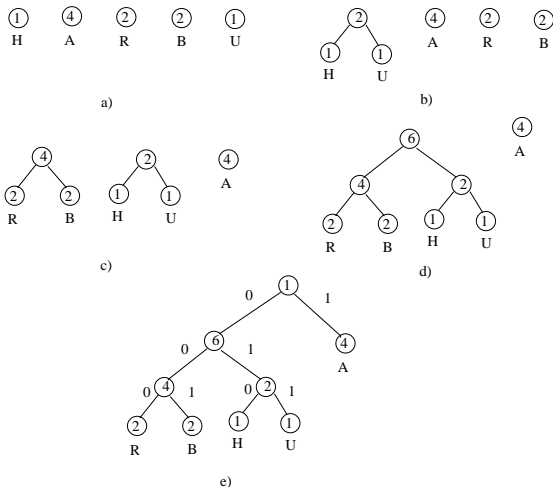
Coduri Huffman - exemplu

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

Literă	Frecvență	Literă	Cod
H	1	H	010
A	4	A	1
R	2	R	000
B	3	B	001
U	1	U	011
a)		b)	

Figura 2 : Codificarea caracterelor din textul HARABABURA

Construcția arborelui Huffman - exemplu



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

Figura 3 : Construcția arborelui Huffman pentru HARABABURA

Algoritm de construcție a arborelui Huffman optim - descriere

- Presupunem că intrarea este memorată într-un tablou T de structuri cu două câmpuri:
 - $T[i].mes$ conține mesajul M_i ;
 - $T[i].f$ conține frecvența f_i .
- Pentru implementare recomandăm reprezentarea arborilor prin tablouri de structuri.
- Notăm cu H tabloul ce reprezintă arborele Huffman.
- Tabloul H conține structuri formate din trei câmpuri:
 - $H[i].elt$;
 - $H[i].fst$ = indicele fiului de pe partea stângă;
 - $H[i].fdr$ = indicele fiului de pe partea dreaptă.
- Semnificația câmpului $H[i].elt$ este următoarea:
 - dacă i este nod intern, atunci $H[i].elt$ reprezintă informația calculată din nod;
 - dacă i este pe frontieră (corespunde unui mesaj), atunci $H[i].elt$ este adresa din T a mesajului corespunzător.
- Notăm cu $val(i)$ funcția care întoarce informația din nodul i , calculată ca mai sus.
- Tabloul H , care în final va memora arborele Huffman corespunzător codurilor optime, va memora pe parcursul construcției acestuia colecțiile intermediare de arbori.

Algoritm de construcție a arborelui Huffman optim - descriere (continuare)

- În timpul execuției algoritmului de construcție a arborelui, H este compus din trei părți (Figura 4):

Partea I: un *min-heap* care va conține rădăcinile arborilor din colecție;

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

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

<i>heap-ul rădăcinilor</i>	noduri care nu nu sunt rădăcini	zonă vidă
----------------------------	------------------------------------	-----------

Figura 4 : Organizarea tabloului H

Algoritm de construcție a arborelui Huffman optim (continuare)

Un pas al algoritmului de construcție ce realizează selecția *greedy* presupune parcurgerea următoarelor etape:

1. Mutarea rădăcinii cu informația cea mai mică pe prima poziție liberă din zona a treia, să zicem k . Aceasta este realizată de următoarele operații:
 - a) copierea rădăcinii de pe prima poziție din heap pe poziția k :
$$H[k] \leftarrow H[1]$$
$$k \leftarrow k + 1$$
 - b) mutarea ultimului element din heap pe prima poziție:
$$H[1] \leftarrow H[m]$$
$$m \leftarrow m - 1$$
 - c) refacerea min-heapului.
 2. Copierea rădăcinii cu informația cea mai mică pe prima poziție liberă din zona a treia, fără a o elimina din min-heap:
$$H[k] \leftarrow H[1]$$
$$k \leftarrow k + 1$$
 3. Construirea noii rădăcini și memorarea acesteia pe prima poziție în min-heap (în locul celei copiate anterior).
 4. Refacerea min-heapului.
- Algoritmul rezultat are timpul de execuție $O(n \log n)$.

Sarcini de lucru și barem de notare

Sarcini de lucru:

1. Dat fiind un text memorat într-un fișier, scrieți un program C/C++ care codifică binar textul astfel încât lungimea medie a codului rezultat să fie minimă (Vezi algoritmul 1ep, figura 4 și algoritmul de construcție a arborelui Huffman optim).

Barem de notare:

1. Implementarea algoritmului algoritmul de construcție 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



Lucanu, D. și Craus, M., *Proiectarea algoritmilor*, Editura Polirom, 2008.



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