Proiectarea algoritmilor

Lucrare de laborator nr. 10

Paradigma *greedy*

Interclasarea optimală

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.*

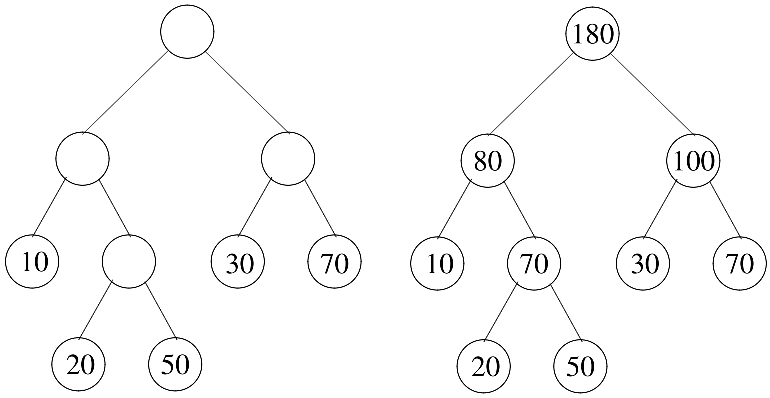


Figura 1: Arbore ponderat pe frontieră, înainte şi după modificare

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 Interclasarea optimală

2.1 Descriere

Se consideră m secvenţe sortate $a_0, \ldots, a_{m-1}$ care conţin $n_0, \ldots, n_{m-1}$, respectiv, elemente dintr-o mulţime total ordonată. Interclasarea celor m secvenţe constă în execuţia repetată a următorului proces:

> Se extrag din mulţime două secvenţe şi se pune în locul lor secvenţa obţinută prin interclasarea acestora.

Procesul se repetă până când se obţine o singură secvenţe sortată cu cele $n_0 + \cdots + n_{m-1}$ elemente.

Problema constă în determinarea unei alegeri pentru care numărul total de transferuri de elemente să fie minim.

Un exemplu este este dat de *sortarea externă*

- Presupunem că avem de sortat un volum mare de date ce nu poate fi încărcat în memoria internă.
- Se partiţionează colecţia de date în în mai multe secvenţe ce pot fi ordonate cu unul dintre algoritmii de sortare internă.
- Secvenţele sortate sunt memorate în fişiere pe suport extern.
- Sortarea întregii colecţii se face prin interclasarea fişierelor ce memorează secvenţele sortate.

Considerăm problema interclasării a două secvenţe sortate:

> Fie date două secvenţe sortate $x = (x_0, \ldots, x_{p-1})$ şi $y = (y_0, \ldots, y_{q-1})$ ce conţin elemente dintr-o mulţime total ordonată. Să se construiască o secvenţă sortată $z = (z_0, \ldots, z_{p+q-1})$ care să conţină cele $p+q$ elemente ce apar în x şi y.

Utilizăm notaţia $z = \text{merge}(x, y)$ pentru a nota faptul că z este rezultatul interclasării secvenţelor x şi y. Numărul de comparaţii executate de algoritmul $\text{merge}(x, y)$ este cel mult $p+q-1$, iar numărul de elemente transferate este $p+q$.

Revenim la problema interclasării a m secvenţe. Considerăm un exemplu: Fie $m = 5, n_0 = 20, n_1 = 60, n_2 = 70, n_3 = 40, n_4 = 30$. Un mod de alegere a secvenţelor pentru interclasare este următorul:

$$\left.\begin{array}{l} b_0 = \text{merge}(a_0, a_1) \\ b_1 = \text{merge}(b_0, a_2) \\ b_2 = \text{merge}(a_3, a_4) \\ b\ \ = \text{merge}(b_1, b_2) \end{array}\right. \quad \text{(vezi Figura 7.a)}$$

Numărul de transferuri al acestei soluţii este $(20+60)+(80+70)+(40+30)+(150+70) = 80+150+70+220 = 520$.

Există alegeri mai bune? Răspunsul este afirmativ!!

2.2 Algoritm

Unei alegeri i se poate ataşa un arbore binar în modul următor:

- informaţiile din vârfuri sunt lungimi de secvenţe;
- vârfurile de pe frontieră corespund secvenţelor iniţiale $a_0, \ldots, a_{m-1}$;
- vârfurile interne corespund secvenţelor intermediare.

Se observă uşor că aceştia sunt arbori ponderaţi pe frontieră şi numărul de transferuri de elemente corespunzător unei alegeri este egală cu LEP a arborelui asociat. Aşadar, alegerea optimă corespunde arborelui cu LEP minimă.

Pentru exemplul anterior ($m = 5, n_0 = 20, n_1 = 60, n_2 = 70, n_3 = 40, n_4 = 30.$), soluţia optimă dată de algoritmul greedy este:

$$\begin{aligned} b_0 &= merge(a_0, a_4) \\ b_1 &= merge(a_3, b_0) \\ b_2 &= merge(a_1, a_2) \\ b &= merge(b_1, b_2) \end{aligned} \quad \text{(vezi Figura 7.b)}$$

Numărul de comparaţii este $50 + 90 + 130 + 220 = 490$.

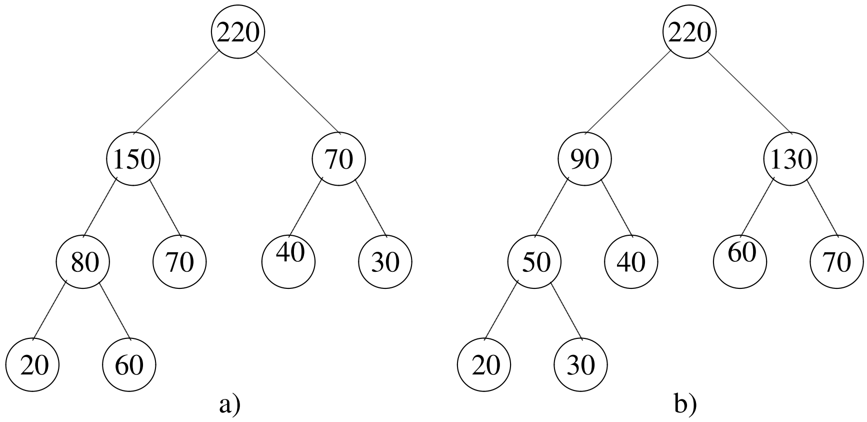


Figura 2: Arborii asociaţi celor doua soluţii de intreclasare

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

1. `T[i].secv` conţine adresa secvenţei sortate a_i;
2. `T[i].n` conţine numărul de elemente din secvenţa a_i.

Algoritmul care urmează utilizează reprezentarea arborilor prin tablouri de structuri.

Notăm cu `H` tabloul ce reprezintă arborele de interclasare. 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:

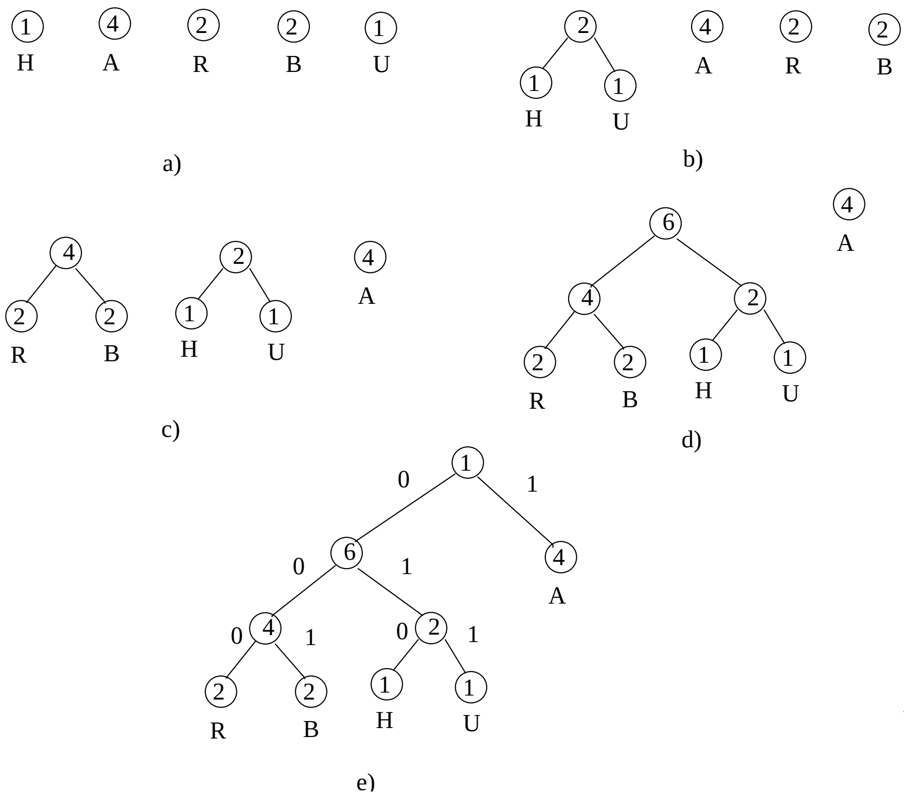
1. dacă `i` este nod intern, atunci `H[i].elt` reprezintă informaţia calculată din nod;
2. dacă `i` este pe frontieră (corespunde unui mesaj), atunci `H[i].elt` este adresa din `T` a secvenţei corespunzătoare.

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;



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

Figura 3: Construcţia arborelui Huffman pentru HARABABURA

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

Figura 4: Organizarea tabloului `H`

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

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

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. Scrieţi o funcţie C/C++ care implementează un algoritm de construcţie a arborelui de interclasare optimală a unei mulţimi de secvenţe sortate.

2. Date fiind n secvenţe sortate, scrieţiun program care să determine o alegere optimă în cazul interclasării a n secvenţe sortate

Barem de notare:

1. Implementarea algoritmului de construcţie a arborelui de interclasare optimală: 6p

2. Determinarea unei alegeri optime în cazul interclasării a n secvenţe sortate: 3p

3. Baza: 1p

Temă suplimentară:

1. Consderăm un graf $G = (V, E)$. Scrieţi un program C/C++ pentru determinarea drumurilor minime între un vârf sursă şi celelalte vărfuri.

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.