Proiectarea algoritmilor

Paradigma *Greedy*

Lucrare de laborator nr. 10

Cuprins

Arbori binari ponderați pe frontieră
- Descriere
- Algoritm pentru construirea unui arbore cu lungimea externă ponderată minimă

Interclasarea optimală
- Descriere
- Algoritm

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 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 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

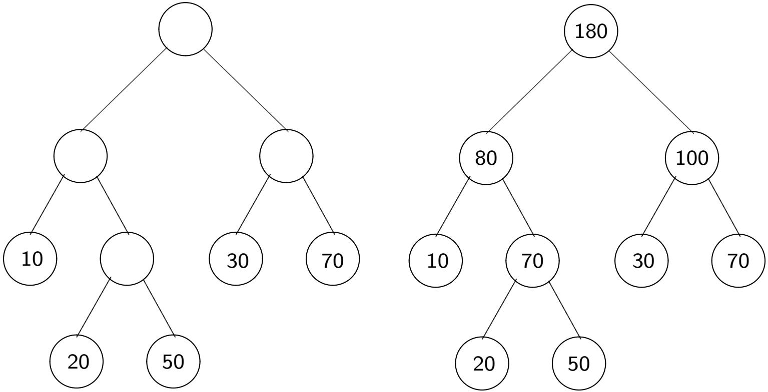

Figura 1: Arbore ponderat pe frontieră, înainte și după modificare

Lungimea externă ponderată

Lemma (1)

Fie t un arbore binar ponderat pe frontieră.
Atunci

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

Lemma (2)

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

Lemma (3)

Presupunem $x_0 \leq x_1 \leq \cdots \leq x_{n-1}$. Există un arbore în $\mathscr{T}(x)$ 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 $x_0 \leq x_1 \leq \cdots \leq x_{n-1}$.
- 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 $\mathrm{LEP}(t1') \leq \mathrm{LEP}(t')$ ce implică

$$\mathrm{LEP}(t1) = \mathrm{LEP}(t1') + x_0 + x_1 \leq \mathrm{LEP}(t') + x_0 + x_1 = \mathrm{LEP}(t)$$

.

- Cum t este optim, rezultă $\mathrm{LEP}(t1) = \mathrm{LEP}(t)$ 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.
- *Notații:* $t(x_i)$ 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 $x_i, i = 0, \ldots, n-1$.

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

Interclasarea optimală a unei mulțimi de secvențe sortate
Descrierea problemei

- Se consideră m secvențe sortate $a_0, \ldots, a_{m-1}$ care conțin $n_0, \ldots, n_{m-1}$, respectiv, elemente dintr-o mulțime total ordonată.
- Interclasarea celor m secvențe constă în execuția repetată a următorului proces:
  - Se extrag din mulțime două secvențe și se pune în locul lor secvența obținută prin interclasarea acestora.
- Procesul se continuă până când se obține o singură secvențe 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 partiționează colecția de date în în mai multe secvențe ce pot fi ordonate cu unul dintre algoritmii de sortare internă.
  - Secvențele sortate sunt memorate în fișiere pe suport extern.
  - Sortarea întregii colecții se face prin interclasarea fișierelor ce memorează secvențele sortate.

Interclasarea unei mulțimi de secvențe sortate - exemplu

- Considerăm problema interclasării a două secvențe sortate:

  Fie date două secvențe sortate $x = (x_0, \ldots, x_{p-1})$ și $y = (y_0, \ldots, y_{q-1})$ ce conțin elemente dintr-o mulțime total ordonată. Să se construiască o secvență sortată $z = (z_0, \ldots, z_{p+q-1})$ care să conțină cele $p+q$ elemente ce apar în x și y.

- Utilizăm notația $z = \mathrm{merge}(x, y)$ pentru a nota faptul că z este rezultatul interclasării secvențelor x și y.
- Numărul de comparații executate de algoritmul $\mathrm{merge}(x, y)$ este cel mult $p+q-1$, iar numărul de elemente transferate este $p+q$.
- Revenim la problema interclasării a m secvențe.
  - 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 secvențelor pentru interclasare este următorul:

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

  - Numărul de transferuri al acestei soluții este $(20+60)+(80+70)+(40+30)+(150+70) = 80+150+70+220 = 520$.
  - Există alegeri mai bune?
  - Răspunsul este afirmativ!!

Interclasarea optimală a unei mulțimi de secvențe sortate - algoritm

- Unei alegeri i se poate atașa un arbore binar în modul următor:
  - informațiile din vârfuri sunt lungimi de secvențe;
  - vârfurile de pe frontieră corespund secvențelor inițiale $a_0, \ldots, a_{m-1}$;
  - vârfurile interne corespund secvențelor intermediare.
- Se observă ușor că aceștia sunt arbori ponderați pe frontieră și numărul de transferuri de elemente corespunzător unei alegeri este egală cu LEP a arborelui asociat.
- Așadar, alegerea optimă corespunde arborelui cu LEP minimă.
- Pentru exemplul anterior $(m = 5, n_0 = 20, n_1 = 60, n_2 = 70, n_3 = 40, n_4 = 30.)$, soluția optimă dată de algoritmul greedy este:

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

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

Arborii asociați celor doua moduri de alegere a secvențelor pentru interclasare

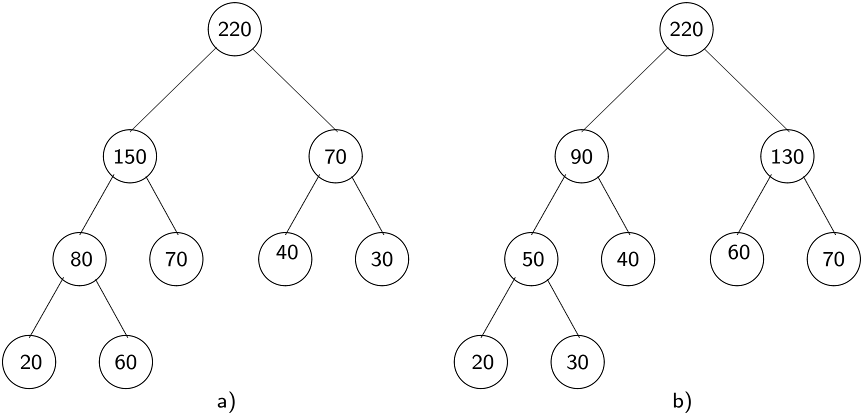

Figura 2: Arborii asociați celor doua soluții de intreclasare

Algoritm de construcție a arborelui de interclasare optimală a unei mulțimi de secvențe sortate - descriere

- Presupunem că intrarea este memorată într-un tablou `T` de structuri cu două câmpuri:
  - `T[i].secv` conține adresa secvenței sortate a_i;
  - `T[i].n` conține numărul de elemente din secvența a_i.
- Algoritmul care urmează utilizează reprezentarea arborilor prin tablouri de structuri.
- Notăm cu `H` tabloul ce reprezintă arborele de interclasare.
- 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 secvenței corespunzătoare.
- Notăm cu `val(i)` funcția care intoarce informația din nodul `i`, calculată ca mai sus.
- Tabloul `H`, care în final va memora arborele de interclasare optimală, va memora pe parcursul construcției acestuia colecțiile intermediare de arbori.

Algoritm de construcție a arborelui de interclasare optimală a unei mulțimi de secvențe sortate - descriere (continuare)

- În timpul execuției algoritmului de construcție a arborelui, `H` este compus din trei părți (Figura 3):

  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.

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

Figura 3: Organizarea tabloului `H`

Algoritm de construcție a arborelui de interclasare optimală a unei mulțimi de secvențe sortate (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] ← H[1]
      k ← k + 1
      ```
   b) mutarea ultimului element din heap pe prima poziție:
      ```
      H[1] ← H[m]
      m ← 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] ← H[1]
   k ← 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. Scrieți o funcție C/C++ care implementează un algoritm de construcție a arborelui de interclasare optimală a unei mulțimi de secvențe sortate.
2. Date fiind *n* secvențe sortate, scrieţiun program care să determine o alegere optimă în cazul interclasării a *n* secvențe sortate.

Barem de notare:

1. Implementarea algoritmului de construcție a arborelui de interclasare optimală: 6p
2. Determinarea unei alegeri optime în cazul interclasării a *n* secvențe sortate: 3p
3. Baza: 1p

Temă suplimentară:

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

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.