

Proiectarea algoritmilor

Căutarea în B-arbori

Lucrare de laborator nr. 4

Cuprins

B-arbori

- B-arborii și indexarea multistratificată

- Crearea B-arborilor

- Căutarea în B-arbori

Link-uri utile

Sarcini de lucru și barem de notare

Bibliografie



B-arborii și indexarea multistratificată

- B-arborii sunt frecvent utilizați la indexarea unei colecții de date.
- Un index ordonat este un fișier secvențial.
- Pe măsură ce dimensiunea indexului crește, cresc și dificultățile de administrare.
- Soluția este construirea unui index cu mai multe niveluri, iar instrumentele sunt B-arborii.
- Algoritmii de căutare într-un index nestratificat nu pot depăși performanța $O(\log_2 n)$ intrări/ieșiri.
- Indexarea multistratificată are ca rezultat algoritmi de căutare de ordinul $O(\log_d n)$ intrări/ieșiri, unde d este dimensiunea elementului indexului.

Index multistratificat

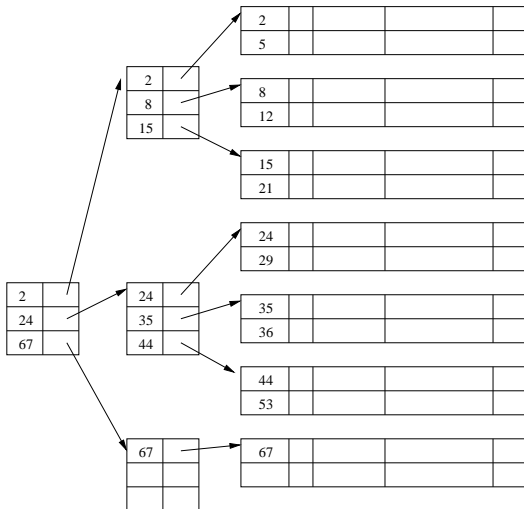


Figura 1 : Index multistratificat organizat pe 3 niveluri



B-arbori - definiție

- Un B-arbore este un arbore cu rădăcină, în care:
 1. fiecare nod intern are un număr variabil de chei și fii;
 2. cheile dintr-un nod intern sunt memorate în ordine crescătoare;
 3. fiecare cheie dintr-un nod intern are asociat un fiu care este rădăcina unui subarbore ce conține toate nodurile cu chei mai mici sau egale cu cheia respectivă dar mai mari decât cheia precedentă;
 4. fiecare nod intern are un fiu extrem-dreapta, care este rădăcina unui subarbore ce conține toate nodurile cu chei mai mari decât oricare cheie din nod;
 5. fiecare nod intern are cel puțin un număr de $f - 1$ chei (f fii), $f =$ factorul de minimizare;
 6. doar rădăcina poate avea mai puțin de $f - 1$ chei (f fii);
 7. fiecare nod intern are cel mult $2f - 1$ chei ($2f$ fii);
 8. lungimea oricărui drum de la rădăcină la o frunză este aceeași.
- Dacă fiecare nod necesită accesarea discului, atunci B-arborii vor necesita un număr minim de astfel de accesări.
- Factorul de minimizare va fi ales astfel încât dimensiunea unui nod să corespundă unui multiplu de blocuri ale dispozitivului de memorare. Această alegere optimizează accesarea discului.
- Înălțimea h unui B-arbore cu n noduri și $f > 1$ satisface relația $h \leq \log_f \frac{n+1}{2}$.



Implementare B-arbori

- Un nod v al unui B-arbore poate fi implementat cu o structură statică formată din câmpurile `tipNod`, `nrChei`, `cheie[nrChei]`, `data[nrChei]` și `fiu[nrChei+1]`.
 - Câmpul `tipNod` conține o valoare $tipNod \in \{frunza, interior\}$.
 - Câmpul `nrChei` conține numărul t al cheilor din nodul v .
 - Tabloul `cheie[nrChei]` conține valorile cheilor memorate în nod (k_0, k_1, \dots, k_{t-1}).
 - Tabloul `data[nrChei]` conține pointerii la structurile care conțin datele asociate nodului v (q_0, q_1, \dots, q_{t-1}).
 - Tabloul `fiu[nrChei+1]` conține pointerii la structurile care implementează fiii nodului v (p_0, p_1, \dots, p_t).

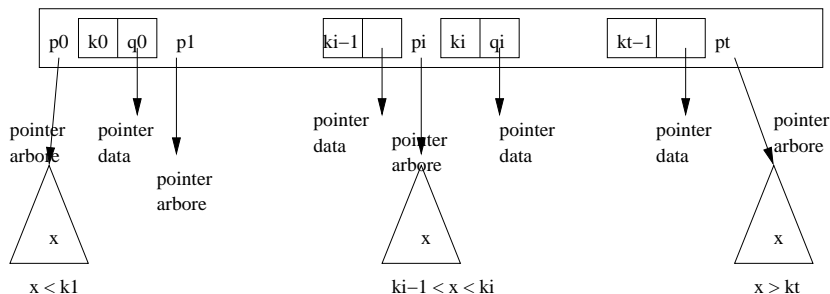


Figura 2 : Structura unui nod al unui B-arbore



Creare B-arbori

- Operația `creeazaBarbore` creează un B-arbore vid cu rădăcina t fără fii.
- Timpul de execuție este $O(1)$.

```
creeazaBarbore(t)
  new t
  t->tipNod ← frunza
  t->nrChei ← 0
  scrieMemorieExterna(t)
end
```



Spargere nod B-arbore - descriere

- Dacă un nod v este încărcat la maxim ($2f - 1$ chei), pentru a insera o cheie nouă este necesară spargerea acestuia.
- Prin spargere, cheia mediană a nodului v este mutată în părintele u al acestuia (v este al i -lea fiu).
- Este creat un nou nod w și toate cheile din v situate la dreapta cheii mediane sunt mutate în w .
- Cheile din v situate la stânga cheii mediane rămân în v .
- Nodul nou w devine fiu imediat la dreapta cheii mediane care a fost mutată în părintele u , iar v devine fiu imediat la stânga cheii mediane care a fost mutată în părintele u .
- Spargerea transformă nodul cu $2f - 1$ chei în două noduri cu $f - 1$ chei (o cheie este mutată în părinte).
- Timpul de execuție este $O(t)$ unde $t = \text{constant}$.



Spargere nod B-arbore - pseudocod

```
procedure spargeNod(u, i, v)
    new nod w
    w->tipNod ← v->tipNod
    w->nrChei ← f-1
    for j ← 0 to f-2 do w->cheie[j] ← v->cheie[j+f]
    if (v->tipNod = interior)
        then for j ← 0 to f-1 do w->fiu[j] ← v->fiu[j+f]
    v->nrChei ← f-1
    for j ← u->nrChei downto i+1 do u->fiu[j+1] ← u->fiu[j]
    u->fiu[i+1] ← w
    for j ← u->nrChei-1 downto i do u->cheie[j+1] ← u->cheie[j]
    u->cheie[i] ← v->cheie[f-1]
    u->nrChei ← u->nrChei + 1
    scrieMemorieExterna(u)
    scrieMemorieExterna(v)
    scrieMemorieExterna(w)
end
```



Inserare în nod incomplet - pseudocod

```
procedure insereazainNodIncomplet(v, k)
  i ← v->nrChei-1
  if (v->tipNod = frunza)
    then while (i >= 0 and k < v->cheie[i]) do
      v->cheie[i+1] ← v->cheie[i]
      i ← i-1
      v->cheie[i+1] ← k
      v->nrChei ← v->nrChei + 1
      scrieMemorieExterna(v)
    else while (i >= 0 and k < v->cheie[i]) do i ← i-1
      i ← i+1
      citesteMemorieExterna(v->fiu[i])
      if (v->fiu[i]->nrChei = 2f-1)
        then spargeNod(v, i, v->fiu[i])
          if (k > v->cheie[i]) then i ← i+1
      insereazainNodIncomplet(v->fiu[i], k)
  end
```



Inserare în B-arbore - descriere

- Pentru a efectua o inserție într-un B-arbore trebuie întâi găsit nodul în care urmează a se face inserția.
- Pentru a găsi nodul în care urmează a se face inserția, se aplică un algoritm similar cu `cautareBarbore`.
- Apoi cheia urmează a fi inserată.
 1. Dacă nodul determinat anterior conține mai puțin de $2f - 1$ chei, se face inserarea în nodul respectiv.
 2. Dacă acest nod conține $2f - 1$ chei, urmează spargerea acestuia.
 3. Procesul de spargere poate continua până în rădăcină.
- Pentru a evita două citiri de pe disc ale aceluiași nod, algoritmul sparge fiecare nod plin ($2f - 1$ chei) întâlnit la parcurgerea *top-down* în procesul de căutare a nodului în care urmează a se face inserarea.
- Timpul de spargere a unui nod este $O(f)$. Rezultă pentru inserție complexitatea timp $O(f \log n)$.



Inserare în B-arbore - pseudocod

```
procedure insereazaBarbore(t, k)
  v ← t
  if (v->nrChei = 2f-1)
    then new nod u
         t ← u
         u->tipNod ← interior
         u->nrChei ← 0
         u->fiu[0]= v
         spargeNod(u, 0, v)
         insereazainNodIncomplet(u, k)
    else insereazainNodIncomplet(v, k)
end
```



Căutare în B-arbori

- Căutarea într-un B-arbore este asemănătoare cu căutarea într-un arbore binar.
- Deoarece timpul de căutare depinde de adâncimea arborelui, `căutareBarbore` are timpul de execuție $O(\log_f n)$.

```
cautareBarbore(v, k)
    i ← 0
    while (i < v->nrChei and k > v->cheie[i]) do i ← i+1
    if (i < v->nrChei and k = v->cheie[i]) then return (v, i)
    if (v->tipNod = frunza)
        then return NULL
        else citesteMemorieExterna(v->fiu[i])
            return cautareBarbore(v->fiu[i], k)
end
```



Link-uri utile

<http://www.bluerwhite.org/btree/>

<http://cis.stvincent.edu/html/tutorials/swd/btree/btree.html>

Sarcini de lucru și barem de notare

Sarcini de lucru: Implementați următoarele operații pentru structuri de tip B-Arbore:

1. Scrieți o funcție C/C++ care implementează operația de inserare a unui element într-un B-Arbore.
2. Scrieți o funcție C/C++ care implementează operația de parcurgere a unui B-Arbore.
3. Scrieți o funcție C/C++ care implementează operația de căutare a unui element într-un B-Arbore.

Barem de notare:

1. Funcția C/C++ care implementează operația de inserare a unui element într-un B-Arbore: 5p
2. Funcția C/C++ care implementează operația de parcurgere a unui B-Arbore: 2p
3. Funcția C/C++ care implementează de căutare a unui element într-un B-Arbore: 2p
4. Baza: 1p

Bibliografie



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