Grupa 143

Proiect Laborator Structuri de Date

5 mai 2021

Echipă formată din:
- Baciu Daniel-Mihai
- Broscoțeanu Daria-Mihaela
- Gherghescu Andreea-Diana

Structura de Date Implementată : **TREAP (Tree + Heap)**

Un treap este un arbore binar care păstrează în același timp atât proprietatea de arbore binar de căutare, cât și cea de max-heap(sau min-heap).

1. **Analiza în timp a programului**
   Pentru o analiză în timp a programului, vom realiza mai întâi o scurtă prezentare a treap-ului și a operațiilor pe care le-am implementat.

   Fiecare nod din cadrul unui treap conține:
- cheie - adică valoarea practic pe care o are nodul - aceasta respectă ordinea din arborele binar de căutare(stânga e mai mică și dreapta e mai mare)
- prioritate - un număr atribuit random care respectă proprietatea de max-heap

În cadrul acestei structuri de date se folosesc rotații pentru a se păstra proprietatea de max-heap în cadrul inserării și ștergerii.

n = numărul de noduri din treap
Operații într-un treap:
- căutarea unui element
  - se realizează identic cu o căutare într-un arbore binar de căutare clasic
  - complexitatea pe cel mai rău caz: O(log n) - întrucât înalțimea maximă este de 1,44*log n

- inserarea unui element
    - se crează un nou nod cu cheia egală cu valoarea de inserat și cu prioritatea egală cu un număr random generat
    - se apelează la o inserare clasică în arbore binar de căutare
    - se utilizează rotații pentru a ne asigura că prioritatea nodului inserat respectă proprietatea de max-heap
    - complexitatea pe cel mai rău caz: O(log n) - întrucât înalțimea maximă este de 1,44*log n

- ștergerea unui element
    - dacă nodul de șters este o frunză, este șters direct
    - altfel se modifică prioritatea nodului la un număr foarte mic și se apelează la rotații pentru a transforma nodul într-o frunză
    -
    - complexitatea pe cel mai rău caz: O(log n) - întrucât înalțimea maximă este de 1,44*log n

- determinarea minimului
    - complexitatea pe cel mai rău caz: O(log n) - întrucât înalțimea maximă este de 1,44*log n
    - parcurgem arborele de la rădăcină până la cea mai din stânga frunză
    - înălţimea maximă a arborelui binar este log n, de unde rezultă complexitatea pe cel mai rău caz
- determinarea maximului
    - complexitatea pe cel mai rău caz: O(log n) - întrucât înalțimea maximă este de 1,44*log n
    - parcurgem arborele de la rădăcină până la cea mai din dreapta frunză
    - înălţimea maximă a arborelui binar este log n, de unde rezultă complexitatea pe cel mai rău caz
- determinarea succesorului
    - complexitatea pe cel mai rău caz: O(log n) - întrucât înalțimea maximă este de 1,44*log n

- determinarea predecesorului
    - complexitatea pe cel mai rău caz: O(log n) - întrucât înalțimea maximă este de 1,44*log n
- determinarea cardinalului
    - aceasta se realizează în O(1) întrucât ne folosim de o variabila pe care o incremenam la inserare și o decrementăm la ștergere
- determinarea numărului de pe poziția k în șirul sortat
    - complexitatea pe cel mai rău caz: O(n)
    - parcurge nodurile arborelui ”în ordine” și contorizează numărul de noduri vizitate până ajunge la cel de pe poziția k
- rotații
    - realizează operațiile de rotație pentru inserare

Astfel, majoritatea operațiilor pe treap se realizează pe cazul cel mai rău în log n, excepție făcând determinarea numărului de pe poziția k în șirul sortat.

Datorită funcțiilor “k_element”, complexitatea timp totală a structurii de date, pentru apelarea fiecărei funcții o singură dată, este O(n).

2. Motivația structurii de date folosite

După o căutare amănunțită, am ales să optăm pentru treap din dorința de a alege o structură de date care să fie cât mai eficientă pentru funcțiile pe care le aveam de implementat.

3. Avantaje/Dezavantaje ale structurii

Avantaje:

- permit implementări ușoare și eficiente ale funcțiilor de inserare, ștergere și căutare
- dacă prioritățile sunt generate random, balansarea arborelui poate fi păstrată cu o foarte mare șansă de reușită
- odată ce rotațiile sunt implementate, restul operațiilor se implementează ușor, ștergerea este chiar mai ușoară decât la un arbore binar de căutare clasic

- e o structură de date cu randomizare, având o mare probabilitate de reușită pe cazuri

Dezavantaje:

- treap-urile, la fel ca arborii binari de căutare pot deveni foarte nebalansați
- nu pot menține proprietăți stricte precum cele de la AVL sau red-black tree
- proprietatea de operații în log n este așteptată doar cu o mare probabilitate

4. Descrierea modului de testare

Corectitudinea și eficiența programului nostru se poate dovedi prin teste de input generate aleator. Programul nostru funcționează cu până la 110.000 de numere în input, rulând corect și pe teste mai mici.

Pentru generarea aleatorie a testelor am optat pentru un program de generare construit de noi care va genera intr-un fișier un model de input care are la început un număr de inserări, apoi interogări.

În cazul nostru, operațiile sunt codificate altfel:

- 0 x - inserează numărul x în treap
- 1 x - șterge numărul x din treap
- 2 - calculează minimul
- 3 - calculează maximul
- 4 x - returnează succesorul nodului x
- 5 x - returnează predecesorul nodului x
- 6 x - returnează numărul de pe poziția x în șirul sortat
- 7 - returnează cardinalul
- 8 x - verifică dacă numărul x se află în treap

în urma rulării se observă că programul nostru funcționează pe cele 8 seturi de teste, obținându-se un timp mult mai bun decât în cazul arborilor binari de căutare clasici. Astfel, rezultatele obținute pe aceste teste sunt:

- pentru testul cu 61 de numere(51 inserări, 10 interogări): 0.026 s
- pentru testul cu 5501 numere(5.001 inserări, 500 interogări): 0.034 s
- pentru testul cu 6001 numere(5.000 inserări, 1.000 interogări): 0.039 s

- pentru testul cu 60001 numere(50.001 inserări, 10.000 interogări): 0.512 s
- pentru testul cu 105.000 numere(100.000 inserări, 5.000 interogări): 0.445 s
- pentru testul cu 110.000 numere(100.000 inserări, 10.000 interogări): 0.724 s
- pentru testul cu 1.100.000 numere(1.000.000 inserări, 100.000 interogări): 4.802 s
- pentru testul cu 11.000.000 numere(10.000.000 inserări, 1.000.000 interogări): 26.025 s

Astfel, obținând rezultate bune pe teste cu mai mult de 100.000 numere, structura de date aleasă se dovedește a fi mai eficientă, atât la nivelul timpului de executare, cât și a implementării în cod a operațiilor, decât un arbore binar de căutare clasic.

5. Sales Pitch

A lucra cu arbori binari de căutare va fii întotdeauna o adevărată încercare pentru cei care vor să stăpânească acest domeniu. Dezechilibrul acestor arbori este o piedică pentru programatorii de pretutindeni.

V-ați săturat să lucrați cu arbori binari nebalansați? V-ați săturat să efectuați operații în timp mai mare decât O(log n)? V-ați săturat să nu lucrați cu structuri de date mai eficiente?

Noi avem soluția!

Treap-ul, un arbore binar de căutare care respectă proprietatea de max-heap, este cheia rezolvării problemelor dumneavoastră!

Eficiența sa, efectuarea majorității operațiilor în O(log n) - n fiind numărul de noduri, ușurința cu care se implementează aceste funcții, ingeniozitatea rotirilor care determină prioritățile arborelui să respecte proprietatea de max-heap și șansa foarte mare ca arborele să rămână balansat sunt lucrurile care fac din treap soluția pentru problema dumneavoastră.

Lucrul cu structuri de date ineficiente se oprește aici! Treap-ul va fii soluția pentru arborele problemelor dumneavoastră! Optați acum pentru TREAP!