Tema 1
Analiza Algoritmilor
Structuri de Date
Mulțimi

Teodora Stroe

321CA
Facultatea de Automatică și Calculatoare
Universitatea Politehnica din București
teodora.stroe2210@stud.acs.upb.ro

1 Introducere

1.1 Descrierea problemei rezolvate

În informatică și inginerie, un set (o mulțime) este un tip abstract de date care poate stoca valori unice, fiind o implementare computerizată a conceptului matematic al unei mulțimi finite.

Unele structuri de date de tip set sunt *statice*, neputând fi modificate după ce au fost construite. Acestea permit numai operații de interogare asupra elementelor lor - cum ar fi verificarea dacă o anumită valoare este în set sau enumerarea valorilor într-o ordine arbitrară. Alte variante, numite mulțimi *dinamice*, permit și inserarea și ștergerea elementelor din mulțime.

De obicei, structura internă a elementelor stocate în aceste structuri de date se află sub formă de perechi *cheie-valoare*. În multiple implementări ale seturilor, care utilizează diferite structuri de date, valoarea unui element este în același timp și cheia acestuia, care îl identifică în mod unic. Acest lucru se poate realiza fie prin încărcarea acesteia cu același conținut ca al cheii, fie prin încărcarea câmpului destinat valorii cu un conținut fictiv, care este ignorat.

În acest articol se analizează două tipuri de structuri de date ce sunt utilizate pentru implementarea mulțimilor în diferite limbaje de programare.

1.2 Exemple de aplicații practice pentru problema aleasă

O generalizare a noțiunii de set este aceea de *multiset*, care este similară cu un set, dar permite valori repetate, "egale". Acest termen este folosit în două sensuri distincte: fie valorile egale sunt considerate identice și sunt pur și simplu numărate, fie valorile egale sunt considerate echivalente și sunt stocate ca elemente distincte.

Un exemplu în acest sens îl poate reprezenta următoarea situație: considerând o listă de mașini, caracterizate de marcă (ex: BMW, Volkswagen, Ford) și anul de fabricație, ar fi posibilă construirea unui multiset de ani, care să calculeze

numărul mașinilor fabricate într-un anumit an. Alternativ, se poate construi un multiset de mașini, în care două vehicule sunt considerate echivalente dacă anii lor de fabricație sunt aceiași. În acest caz, pot exista mașini diferite, cu mărci diferite, caz în care fiecare pereche (marcă, an de fabricație) trebuie să fie stocată separat.

În bazele de date relaționale, înregistrările unui tabel pot reprezenta un set sau un multiset, în funcție de eventualele constrângeri de unicitate.

2 Specificarea soluțiilor alese

Două dintre metodele principale de implementare ale mulțimilor sunt prin:

1. tabele de dispersie (C++11: unordered_set; STL[1]: hash_set; Java: HashSet)
2. arbori binari de căutare echilibrați (C++: set; Java: TreeSet)

În cadrul acestui articol se realizează o analiză comparativă a acestor structuri de date, și anume tabele de dispersie (hashtable-uri) versus arbori binari de căutare echilibrați, în cazul acestora din urmă efectuându-se o discuție pe trei cazuri particulare: AVL, Treap-uri și R&B Trees.

Din punct de vedere al implementării, pentru fiecare dintre cele două structuri, se vor pune în discuție operațiile de inserare, ștergere, respectiv modificare a unui element. În ceea ce privește operația de modificare, aceasta va reprezenta o compunere a operațiilor de ștergere și inserare. Această abordare este aleasă pentru a imita cât mai bine comportamentul set-urilor actuale din Java.

De asemenea, datele reținute în câmpul de valoare vor avea același conținut cu cheia, rezultând atât o uniformizare a formatului datelor de intrare, cât și o simulare a actualelor implementări.

3 Criterii de evaluare

În realizarea fișierelor utilizate pentru evaluarea algoritmilor menționați anterior, se vor lua în considerare următoarele aspecte:

- tipul de date conținute în test;
- dimensiunea datelor testului;
- rolul testului.

După criteriul dimensiunii și al tipului de date, se vor realiza două categorii de fișiere de intrare, cu date de tip întreg (int) și șiruri de caractere (char), de mici dimensiuni pentru a testa corectitudinea, și de dimensiuni mari, pentru a testa complexitatea și pentru a genera o analiză comparativă între cele două tipuri de structuri.

De asemenea, vor fi create teste pentru verificarea cazurilor limită și a situațiilor speciale, în funcție de structura de date analizată (ex: hashtable - coliziuni, modificarea/ștergerea unei valori inexistente).

[1] The SGI Standard Template Library

Bibliografie

1. T. Cormen, C. Leiserson, R. Rivest and C. Stein, Introduction to algorithms, 3rd ed. Massachusetts Institute of Technology, 2009, p. 1313.
2. D. Knuth, The art of computer programming, vol 3, 2nd ed. Reading, Mass.: Addison-Wesley, 1997, p. 803
3. D. Liu and S. Xu, "Comparison of Hash Table Performance with Open Addressing and Closed Addressing: An Empirical Study", International Journal of Networked and Distributed Computing, vol. 3, no. 1, p. 60, 2015
4. T. Wang, "Sorted Linear Hash Table", 1997 [Online]
5. T. Wang, "Integer Hash Function", 1997 [Online]
6. Documentație disponibilă la `www.cplusplus.com`