Tema Analiza Algoritmilor

Etapa Finala

Roșu Mihai Cosmin 323CA

Facultatea de Automatica si Calculatoare
Universitatea POLITEHNICA Bucuresti
mihaicosminrosu@yahoo.com

Abstract. Aceasta tema analizeaza din punct de vedere al eficientei de timp tabelele de dispersie(Hashtable) si arborii binari de cautare echilibrati(Treap). Acestea vor fi comparate pe baza mai multor teste in urma carora se vor determina modurile in care se comporta aceste structuri de date in diferite situatii.

Keywords: Structuri de date · Tabele de dispersie · Arbori binari de cautare echilibrati · Treap

1 Introducere

1.1 Descrierea problemei rezolvate

In zilele noastre, cantitatea de date creste intr-un ritm alarmant, iar metodele de stocare a acestora devin tot mai importante. Nu este de ajuns ca acestea sa fie stocate pur si simplu intr-un array unidimensional sau bidimensional, intrucat, de cele mai multe ori, aceste date trebuie modificate, sterse, sau chiar aranjate intr-o anumita ordine, asa ca este necesar un mod de stocare in care aceste operatii sa fie cat mai eficiente posibil. De asemenea, trebuie luata in calcul cantitatea de date. Pentru un numar mic de date nu este foarte relevanta structura de date folosita, insa la cantitati mari structura de date aleasa poate face diferenta intre cateva secunde si cateva ore, zile sau chiar ani.

1.2 Aplicatie practica la problema

Datorita fenomenului digitalizarii tot mai multe companii trec in mediul online si au nevoie de baze de date. Totodata, este necesar ca accesarea acestor baze de date sa fie una rapida, la un simplu click pe site-ul companiei. Din aceasta cauza, modul in care sunt stocate datele este foarte important pentru o companie, asa ca trebuie aleasa structura de date optima pentru ca viitorii clienti sa aiba o experienta cat mai buna.

1.3 Specificarea solutiilor alese

Pentru rezolvarea problemei am ales tabelele de dispersie (Hashtables) si arborii binari de cautare echilibrati (Treap). Tabelele de dispersie retin perechi (cheie, valoare), iar Treap-ul retine valori simple. Eficienta celor doua este obtinuta in moduri diferite. In cazul tabelelor de dispersie, eficienta se obtine prin folosirea unei functii de hash pentru ca datele sa fie stocate cat mai uniform, iar in cazul Treap-ului eficienta se obtine prin modul stocarii, astfel incat printr-o parcurgere inordine sa se obtina datele sortate crescator, iar faptul ca arborele este echilibrat duce la evitarea cazului in care structura devine o lista simplu inlantuita.

1.4 Criteriile de evaluare pentru solutia propusa

Mai intai, voi testa functionalitatea operatiilor implementate pe teste de dimensiune mica, iar apoi voi testa eficienta structurilor pe teste de dimensiune mare. Testele vor respecta o structura simpla: pe fiecare rand se va gasi o comanda. Pentru simplitate datele ce trebuie retinute vor fi numere intregi (de tip int), iar in cazul tabelelor de dispersie, unde sunt necesare perechi (cheie, valoare), cheia se va considera egala cu valoarea. O comanda este formata dintr-un caracter urmat de unul, doua sau zero numere intregi, separate printr-un spatiu. In functie de caracter se va alege operatia. Exista cinci operatii posibile: add (ex: A 5), remove (ex: R 5), modify (ex: M 5 6), find (ex: F 5) si print (ex: P). Operatia de add va adauga un nou element in structura de date, cea de remove va sterge un element din structura, cea de modify va schimba un element din structura intr-un element nou, cea de find va cauta daca exista elementul in structura si va intoarce 1 daca il gaseste, iar 0 altfel si pentru operatia de print se vor afisa toate elementele stocate in structura.

2 Prezentarea solutiilor

2.1 Descrierea modului in care functioneaza algoritmii alesi

2.1.1 Tabele de dispersie

Aceasta structura de date este compusa dintr-o colectie de chei si o colectie de valori, in care fiecare cheie are asociata cate o valoare, formand perechi (cheie, valoare). La baza functionalitatii sta procedeul de hashing. In cadrul implementarii tabelelor de dispersie este necesara o functie de hash, care primeste cheia si genereaza un indice, folosit pentru a alege locul unde se va stoca (sau unde se va accesa) informatia. Ideal, fiecare cheie ar trebui sa genereze un indice diferit, motiv pentru care este importanta alegerea unei functii bune de hash. Daca doua chei genereaza acelasi indice are loc o coliziune. Tratarea coliziunilor se poate face in doua moduri: inlantuire (cheile care genereaza acelasi indice sunt adaugate intr-o lista) si adresare deschisa (se cauta prima pozitie libera dupa indicele generat). Inlantuirea este mai rapida pentru un numar mare de coliziuni, insa adresarea deschisa este mai eficienta pentru un numar mic si nu necesita alta structura de date (lista). In spatele unui tabel de dispersie se afla un vector (array) care foloseste indicele generat de functia de hash.

2.1.2 Treap

Un arbore binar de cautare este un arbore binar in care valorile apartin unei multimi peste care exista o relatie de ordine totala si fiecare nod este mai mare decat toate nodurile din subarborele stang si mai mic decat toate nodurile din subarborele drept. Treap-ul este un arbore binar de cautare echilibrat. Fiecare nod retine in plus o prioritate care este aleasa aleator la adaugarea nodului in structura. Folosind operatii de rotire, care mentin proprietatile unui arbore binar de cautare, elementele Treap-ului se rearanjeaza astfel incat prioritatile retinute sa indeplineasca proprietatile unui Heap (fiecare nod are prioritatea mai mare sau egala cu proprietatile fiilor). In acest mod se evita cazul in care arborele binar de cautare ajunge sa fie o lista inlantuita.

2.2 Analiza complexitatii solutiilor

2.2.1 Tabele de dispersie

Deoarece in spatele unui tabel de dispersie se afla un vector (array), operatiile de add, remove si modify au complexitatea:

$$O(1) \tag{1}$$

deoarece aceasta este complexitatea accesarii elementelor unui vector, iar functia de hash are timp constant. Totusi exista cazul cand tabelul de dispersie ajunge sa fie plin si sa necesite sa fie redimensionat. Prin urmare pentru operatia de resize, care implica scoaterea tuturor celor n elemente din tabelul vechi si inserarea lor in tabelul nou, de dimensiune dubla, complexitatea este:

$$O((1+1) \cdot n) = O(2 \cdot n) = O(n) \tag{2}$$

unde n este numarul de elemente din tabelul de dispersie.

2.2.2 Treap

Deoarece Treap-ul este un arbore binar echilibrat inseamna ca pentru n elemente arborele va avea inaltimea:

$$h = [logn] \tag{3}$$

Operatiile de add si remove parcurg arborele de la radacina la frunze, iar apoi aplica operatii de rotire, care pornesc de la frunze si, in cel mai rau caz, merg pana la radacina. Pentru operatia de modify se intampla acelasi lucru, dar de doua ori (o data pentru stergerea elementului vechi si o data pentru adaugarea celui nou). Prin urmare, complexitatea operatiilor este:

$$O(2 \cdot h) = O(h) = O([logn]) = O(logn) \tag{4}$$

unde n este numarul de elemente din Treap.

2.3 Prezentarea principalelor avantaje si dezavantaje pentru solutiile luate in considerare

2.3.1 Tabele de dispersie

Avantajul acestei structuri este eficienta operatiilor de add, remove si modify. Pe de alta parte, un dezavantaj este faptul ca daca nu se cunoaste numarul de chei ce vor fi adaugate va fi necesara redimensionarea tabelului de dispersie, ceea ce duce la scaderea eficientei. De asemenea, structura necesita o functie buna de hash care sa genereze acelasi indice pentru aceeasi cheie de fiecare data, sa evite coliziunile, sa genereze indici variati pentru chei apropiate si sa nu fie inversate usor. Daca functia de hash nu este buna vom avea un numar mare de coliziuni. Daca tratam coliziunile prin inlantuire vom obtine liste lungi, iar in cazul adresarii deschise va fi necesara parcurgerea unor portiuni mari din vectorul tabelului. Ambele cazuri ar duce la scaderea eficientei, si pierderea complexitatii constante. Totodata, singurul mod prin care se pot afisa toate elementele stocate este prin parcurgerea vectorului din tabel, ceea ce ar duce la obtinerea elementelor intr-o ordine aleatoare care tine de indicii generati de functia de hash.

2.3.2 Treap

Avantajul Treap-ului este ca nu este necesara cunoasterea numarului de elemente ce vor fi inserate pentru performanta maxima. De asemenea, printr-o parcurgere inordine (fiu stanga, radacina, fiu dreapta) se pot obtine elementele sortate cu o complexitate polinomiala. In schimb, un dezavantaj este complexitatea logaritmica a operatiilor de add, remove si modify, care este mult mai slaba decat complexitatea constanta a tabelelor de dispersie.

3 Evaluare

3.1 Descrierea modalitatii de construire a setului de date folosite pentru validare

Pentru simplitate in cadrul testelor am considerat ca structurile vor retine numere intregi, iar in cazul tabelelor de dispersie cheia va fi egala cu valoarea. Primele trei teste sunt folosite pentru a testa functionalitatea operatiilor pe cazuri simple si sunt generate manual. Pentru restul de 24 de teste am folosit un generator scris in C, intrucat testele sunt de 1000, 10000, 100000, respectiv 1000000 de elemente. Cele 24 de teste se incadreaza in 6 categorii: doar add-uri, doar modify-uri (cu mentiunea ca la orice moment dat se afla un singur numar in structuri), doar remove-uri (si add-urile necesare), o combinatie a celor trei operatii (add-uri, apoi moddify-uri, iar la final remove-uri), add-uri urmate de remove-uri si apoi din nou add-uri (scopul acestui tip de test este de a vedea diferenta atunci cand tabelul de dispersie are dimensiunea suficient de mare), respectiv o combinatie a celor trei operatii, in care se folosesc operatii doar pe anumite numere.

3.2 Mentionati specificatiile sistemului de calcul pe care ati rulat testele (procesor, memorie disponibila)

Masina virtuala pe care am rulat testele are urmatoarele specificatii:

- Procesor: Intel(R) Core(TM) i7-6700K CPU @ 4.00GHz
- GPU: NVIDIA GeForce GTX 980
- RAM: 4 GB
- Memorie totala: 20.5 GB
- Memorie folosita de sistem: 14.7 GB
- Memorie disponibila: 5.8 GB
- Sistem de operare: Ubuntu 64-bit

3.3 Ilustrarea, folosind grafice/tabele, a rezultatelor evaluarii solutiilor pe setul de teste

Table 1. Ilustrarea timpilor de rulare in secunde

Numar test	Ordin de marime	Hashtable	Treap
1	10	**0.001**	**0.001**
2	10	**0.001**	**0.001**
3	10	**0.001**	**0.001**
4	1000	**0.002**	**0.002**
5	1000	**0.001**	**0.001**
6	1000	0.002	**0.001**
7	1000	**0.002**	**0.002**
8	1000	**0.002**	**0.002**
9	1000	**0.002**	**0.002**
10	10000	0.01	**0.005**
11	10000	**0.003**	**0.003**
12	10000	0.012	**0.006**
13	10000	**0.017**	**0.017**
14	10000	0.016	**0.01**
15	10000	0.017	**0.016**
16	100000	0.137	**0.093**
17	100000	0.026	**0.02**
18	100000	0.14	**0.092**
19	100000	**0.245**	0.4
20	100000	0.249	**0.2**
21	100000	0.262	**0.242**
22	1000000	**1.776**	6.855
23	1000000	0.238	**0.195**
24	1000000	**1.877**	7.246
25	1000000	**2.895**	47.308
26	1000000	**3.141**	13.385
27	1000000	**3.295**	14.299

Ordinul de marime reprezinta dimensiunea testelor, din punct de vedere al numarului de comenzi, aproximata la cea mai apropiata putere a lui 10.

3.4 Prezentarea, succinta, a valorilor obtinute pe teste. Daca apar valori neasteptate, incercati sa oferiti o explicatie

Datele din tabelul de mai sus ne arata ca pe un numar mic de valori Treap-ul este mai eficient. Insa, pentru un numar mare de valori (> 1000000) tabelul de dispersie este semnificativ mai eficient. Printre valorile neasteptate se regasesc cele de la testele 20 si 25. Pentru testul 20, care se incadreaza in categoria testelor care testau cum se schimba eficienta tabelului de dispersie atunci cand dimensiunea array-ului este indeajuns de mare, tabelul de dispersie este inca mai ineficient decat Treap-ul. In cazul testului 26 valoarea neasteptata este cea a Treap-ului care ruleaza timp de 47 de secunde, mult mai mult decat orice alt test pe oricare dintre cele doua structuri. Pentru restul testelor valorile obtinute sunt conform asteptarilor, intrucat pe baza complexitatilor putem spune ca Treap-ul va ajunge sa fie mai ineficient. Motivul pentru care initial este mai eficient este faptul ca Hashtable-ul incepe cu dimensiunea de un element, dupa care de fiecare data cand ajunge sa fie plin trebuie sa fie redimensionat la o dimensiune dubla. Acest lucru inseamna ca initial vor fi multe redimensionari care au complexitate polinomiala, mai mare decat cea logaritmica a Treap-ului. Insa, la dimensiuni mari redimensionarile ajung sa aiba loc foarte rar, moment in care tabelul de dispersie devine mai eficient.

4 Concluzie

In concluzie, dupa analiza celor doua structuri de date am ajuns la concluzia ca ambele sunt situationale. De exemplu, daca se cunoaste numarul elementelor, tabelele de dispersie sunt mai bune decat Treap-ul, datorita complexitatii constante. Totusi, daca nu se cunoaste acest numar, Treap-ul este mai bun, dar pentru un numar mic de numere (< 1000000). Totodata, ambele au proprietati care sunt unice: tabelele de dispersie pot retine perechi, iar Treap-urile pot sorta datele stocate.

References

1. What are Data Structures?, `https://searchsqlserver.techtarget.com/definition/data-structure`. Ultima accesare 14 Decembrie 2021
2. Real-time application of Data Structures, `https://www.geeksforgeeks.org/real-time-application-of-data-structures/`. Ultima accesare 14 Decembrie 2021
3. Structuri de Date - Laborator 4 - Dictionar, `https://ocw.cs.pub.ro/courses/sd-ca/laboratoare/lab-04`. Ultima accesare 14 Decembrie 2021
4. Structuri de Date - Laborator 10 - Treap, `https://ocw.cs.pub.ro/courses/sd-ca/laboratoare/lab-10`. Ultima accesare 14 Decembrie 2021