ГОДИШНИК на Минно-геоложкия университет "Св. Иван Рилски", Том 54, Св.III, Механизация, електрификация и автоматизация на мините, 2011
ANNUAL of the University of Mining and Geology "St. Ivan Rilski", Vol. 54, Part III, Mechanization, electrification and automation in mines, 2011

ПРЕДСТАВЯНЕ И СОРТИРАНЕ НА ДАННИ ЧРЕЗ ДВОИЧНИ ДЪРВЕТА

***Петко Лалов*[1], *Тихомир Благоев*[2]**

[1]Минно-*геоложки университет "Св. Иван Рилски", 1700-София, e-mail: petko@mgu.bg*
[2] Минно-*геоложки университет "Св. Иван Рилски", 1700-София*

РЕЗЮМЕ. Една от важните задачи при обработката на данни е тяхното представяне и сортиране. В тази статия разглеждаме случая на последователно четене на данни от файл, конзола , стек и др. Показва се как тези несортирани данни могат да бъдат представени като двоично кореново дърво с времева сложност $O(n\log_2 n)$, използвайки процедура близка до метода на Хоор за бързо сортиране, но избягвайки рекурсия. След това даваме линеен алгоритъм за сортиране на данните, разположени във върховете на дървото.

BINARY TREES REPRESENTATION AND SORTING OF DATA

Petko Lalov[1], Tihomir Blagoev[2]
[1]University of Mining and Geology *"St. Ivan Rilski", 1700-Sofia, e-mail: petko@mgu.bg*
[2] University of Mining and Geology *"St. Ivan Rilski", 1700-Sofia*

ABSTRACT An important problem in data processing is their representation and sorting. The current paper looks through the case of serial data retreival from file, console, steak and etc. Is shown how these unsorted data can be represented as a binary root tree with timing complexity of $O(n\log_2 n)$, using avoiding recursion procedure similar to the Hoor method of quick sorting. Then we deliver a linear algorithm for sorting of data situated on the vertices of the tree.

I. Въведение

Процесът на пренареждане на елементите на някакво множество от обекти в определен ред се нарича сортиране. Сфера на приложение: телефонни указатели, речници, справочни индекси и др.

Съществуват различни класификации на алгоритмите за сортиране. Най- популярната е в зависимост от местоположението на данните. Изхождайки от този критерий различаваме **вътрешно**(данните се намират в ОП на компютъра) и **външно** (данните са във външната памет и достъпа до тях се осъществява последователно).

В зависимост от операцията, извършвана над елементите, различаваме сортиране чрез **сравнение**(най- често с операции >, <, =) и чрез **трансформация** (с помощта на аритметични операции).

Нека е дадено множество M с елементи $a_1,a_2,\ldots..,a_n$ и функция f дефинирана върху тях. Под *сортиране* на елементите на M ще разбираме пермутирането им в подходящ ред $a_{i1}, a_{i2},\ldots\ldots,a_{in}$ така, че да бъде изпълнено $f(a_{i1}) \le f(a_{i2}) \le\ldots\ldots\le f(a_{in})$,
където f- се нарича функция на наредба на множеството[Н.Вирт, 1989].

При методите за сортиране най- важните характеристики са времевата сложност и необходимата памет. Най- често данните се представят чрез структурата масив и сортирането се извършва в рамките на този масив чрез операцията сравнение и размяна на местата на елементите на масива. Добре известни са метода на „мехурчето", метод на „вмъкването", метод на Шел и др.Повечето от тях са с времева сложност $O(n^2)$. Най-голям интерес представлява, създаденият през 1962 година "бърз" алгоритъм на Хоор[П. Наков, П. Добриков, 2005], който при прилагане на поредица от евристики може да доведе до времева сложност $O(n \log_2 n)$. Компютърната реализация на този алгоритъм обикновено се извършва чрез рекурсивна процедура. По-долу ще предложим друг алгоритъм, който се основава на разполагането на постъпващите данни във върховете на двоично кореново дърво за претърсване, като чрез него и линейно хеширане могат да бъдат сортирани данните.

I. Алгоритъм

- **Определения**:

Двоично кореново дърво- всеки връх на дървото има 1 или 2 наследници, като върховете са разположени на нива, като нулевото ниво съдържа само един връх, наречен корен(root).

Не е трудно да се прецени, че даден връх v принадлежи на ниво k, ако дължината на пътя от корена до v е k.

Както отбелязахме по- горе всеки връх на двоичното дърво маркираме с конкретна данна, наречена ключ на върха.

Всички наследници на root с ключове по- малки от ключа на root образуват така нареченото **ляво поддърво** на root.

Всички наследници на root с ключове по- големи от ключа на root образуват така нареченото **дясно поддърво** на root.

Следва, че всеки връх генерира ляво и дясно поддърво на този връх.

Двоично дърво за претърсване- ключовете на върховете на лявото поддърво на даден връх са по- малки или равни от ключа на този връх, а ключовете на върховете на дясното поддърво на даден връх са по- големи или равни от ключа на този връх.

- **Генериране на двоичното дърво от данни**

Приемаме, че данните $a_1,a_2,.....,a_n$ се третират по реда на постъпване. За корен на дървото се избира a_1. За всеки елемент a_i се изпълнява следната двоична процедура за намиране на мястото на този елемент във връх на двоичното дърво:

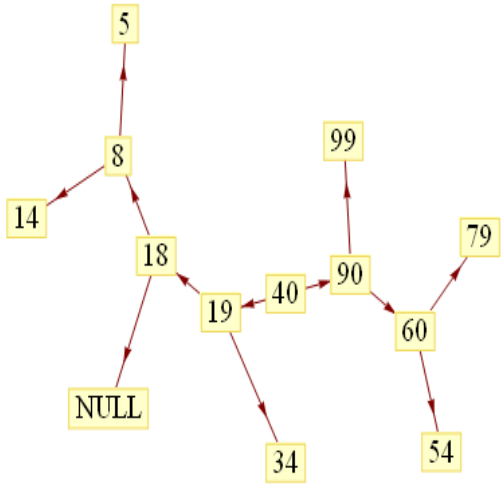
1)ако

$$a_i < root \quad a_i \in Left(root) \quad else \quad a_i \in Rigth(root)$$

2) ако наследниците на root са свободни a_i заема мястото на единия от тях; в противен случай двоичното търсене продължава.

На долната схема сме показали генерирането на двоично дърво за данните:

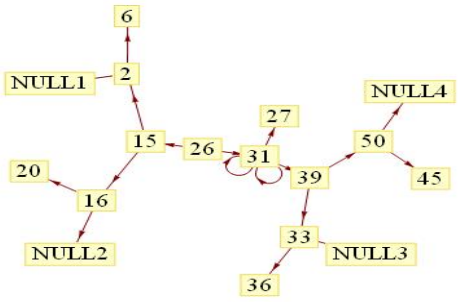
{40,19,18,90,60,54,99,8,5,14,79,34}



Фиг.1

Алгоритъмът работи и при многократно повтаряне на някои данни. В този случай някои дъги на дървото се израждат в дъги с общо начало и край, но този проблем не нарушава единствеността на последващото хеширане. На следващия пример илюстрираме такава ситуация за данните:

{26,31,39,15,50,16,31,27,2,20,33,45,31,36,6}



Фиг.2

- **Хеширане**

След построяването на двоичното дърво с ключове на върховете самите данни трябва да се дефинира хеш-функция, която да съпоставя на всяка данна еднозначно нейният линеен адрес(принадлежащ на {1, 2, …,n}) в сортирания списък.

За всеки връх i, в процеса на генериране на дървото се определят:

$|Left(i)|$ – мощността на лявото поддърво с родител i;

$|Rigth(i)|$ – мощността на дясното поддърво с родител i;

N(i)- хеширания ключ на данната i в сортирания масив;

Стартира се с i=root.

$$N(root) = |Left(root)| + 1;$$

Последователно се определят хешираните ключове на наследниците на root, на техните наследници и т.н.

Ако сме определили хеш- ключовете на върховете $\{V_h\}$ на височина h, то хеш- ключовете на върховете-наследници на височина h+1 се пресмятат по следния начин:

Не е трудно да се прецени, че ако $v \in \{V_h\}$ и v' е ляв наследник на v:

$$N[v'] = N[v] - |Left(v)| + |Left(v')|$$

Ако v' е десен наследник на v:

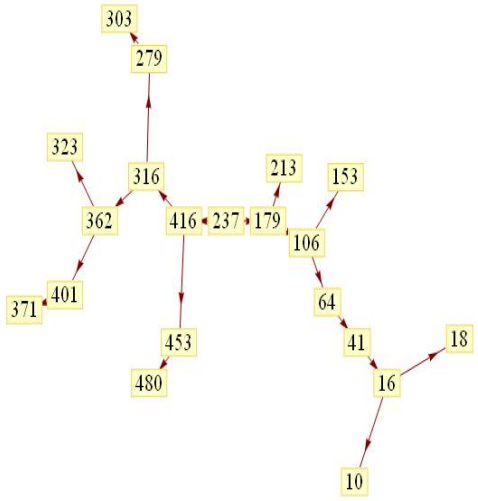
$$N[v'] = N[v] + |Rigth(v)| - |Rigth(v')|$$

Ще илюстрираме алгоритъма с един пример:

Данни:

```
{237, 179, 106, 416, 453, 64, 213, 316, 153,
 362, 41, 16, 480, 401, 10, 279, 18, 303, 323, 371}
```

root=237



Фиг.3

Хеш таблица:

{237 → 10, 179 → 8, 416 → 18, 106 → 6, 213 → 9, 316 → 13, 453 → 19,
64 → 5, 153 → 7, 279 → 11, 362 → 15, 480 → 20, 41 → 4,
303 → 12, 323 → 14, 401 → 17, 16 → 2, 371 → 16, 10 → 1, 18 → 3}

- **Ефективност на алгоритъма**

Предложеният алгоритъм изисква допълнителна памет от порядъка на 2n. По отношение на времевата сложност основна роля играе броят на операцията „сравнение". Ако данните са разположени в дървото на к нива, съответно с брой данни $n_1 \le 2,\ n_2 \le 2^2, \ldots\ldots, n_k \le 2^k$, като броя на сравненията за определяне мястото на данните на ниво I е I, I=1, 2, ….,k. По този начин общият брой сравнения за построяване на дървото е:

$$N = \sum_{i=1}^{k} i n_i \qquad (1)$$

Очевидно, колкото е по- малък броят на нивата, N е по-малко. Най-добрият случай е когато дървото е балансирано, т.е. височините на лявото и дясното поддърво на корена се различават най- много с 1. Да изчислим в този случай броя на сравненията.

Тъй като дървото е балансирано и данните са разположени на k нива, като на ниво 1 има точно 2 данни, на ниво 2- 4 данни,.....на ниво k-1- 2^{k-1} данни. На ниво к в общия случай ще има по малко от 2^k данни, но за улеснение на пресмятанията допуска ме, че са 2^k.

Ако данните са n, броят на нивата k се определя от

$$1 + 2 + 2^2 + \ldots\ldots + 2^k = 2^{k+1} - 1 = n,$$

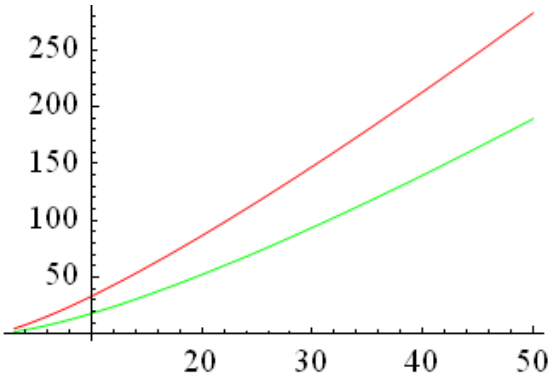
откъдето

$$k = \log_2(n+1) - 1$$

В този случай

$$N_{opt} = (n+1)\log_2(n+1) - 2n \qquad (2)$$

и тази оценка в този благоприятен случай е по- добра от най- добрата оценка на метода на Хоор- $O(n\log_2 n)$, което се вижда от долните графики на двете функции.



Чрез създадената от нас компютърна програма бяха направени множество експерименти, като времевата сложност се изчисляваше от (1) и се сравняваше с най-добрата оценка на Хоор. При случайно генериране на данни нашата оценка по отношение броя на сравненията в повечето случаи беше по- добра.

Литература

Н.Вирт Алгоритмы и структуры данных, Moscow, Мир1989, (in Russian).

П. Наков, П. Добриков Програмиране=++Алгоритми София 2005

Препоръчана за публикуване от катедра „Информатика", МЕМФ