Седма лекция по ДАА на втори поток КН
Сортиране в линейно време. Counting Sort. Radix Sort

14 април 2014

Абстракт

Демонстираме сортиране в линейно време върху вход, чиито елементи не са произволни, а се подчиняват на определени ограничения. Въвеждаме сортиращите алгоритми Counting Sort и Radix Sort

Съдържание

1 Сортиране в линейно време

Както видяхме в предната лекция, сортирането на прозволни елементи, чиито индивидуални стойности не можем четем, а само можем да сравняваме елементите един с друг, не може да става асимптотично по-бързо от $n \lg n$. Но тази долна граница не е в сила, ако има някакви ограничения възможните стойности на елементите. Съвсем прост пример е сортирането на булев масив – очевидно е достатъчно да преброим колко са елементите от всеки вид и после да презапишем входа със съответния брой нули и единици, като нулите са преди единиците, което може да стане в $\Theta(n)$ време. Алгоритъмът, който ще разгледаме сега, се основава на идеята за преброяване на елементите от всяка големина, но както ще стане ясно, той третира елементите от входа не просто като числа, а като записи, чиито ключове са числа. Тъй като записите освен ключовете имат и някаква сателитна информация, сортирането става с местене на елементи (а не просто презаписване на входа с числа). Дори когато казваме "входът е масив от числа", имаме предвид, че входът е масив от записи с ключове-числа.

2 Counting Sort

Входът е масив от числа $A[1, \ldots, n]$. Известно е, че за някакво $k \in \mathbb{N}^+$, за всяко $A[i]$ е изпълнено

$$A[i] \in \{1, 2, \ldots, k\}$$

Алгоритъмът ползва работен масив $C[0, 1, \ldots, k]$ и още един масив $B[1, 2, \ldots, n]$, в който се записва резултата от сортирането.

Counting Sort($A[1,2,\ldots,n]$: positive integers; k: a positive integer)

```
1   (* k е такова, че 1 ≤ A[i] ≤ k за всички i *)
2   for i ← 0 to k
3       C[i] ← 0
4   for i ← 1 to n
5       C[A[i]] ← C[A[i]] + 1
6   for i ← 1 to k
7       C[i] ← C[i] + C[i − 1]
8   for i ← n downto 1
9       B[C[A[i]]] ← A[i]
10      C[A[i]] ← C[A[i]] − 1
```

Първо ще покажем работата на алгоритъма с пример и после ще дадем формално доказателство за коректност. Нека $n = 8$, $k = 6$ и масивът $A[\,]$ е:

A	3	6	4	1	3	4	1	4
	1	2	3	4	5	6	7	8

Очевидно първият **for**-цикъл (редове 2–3) просто нулира масива $C[\,]$. Вторият **for**-цикъл (редове 4–5) преброява по колко елемента от всяко $i \in \{1,\ldots,k\}$ има в $A[\,]$ и записва това в $C[\,]$:

C	0	2	0	2	3	0	1
	0	1	2	3	4	5	6

Очевидно, $\sum_{i=0}^{k} C[i] = n$. Третият **for**-цикъл (редове 6–7) присвоява на всяко $C[i]$ броя на всички елементи на $A[\,]$, по-малки или равни на i:

C	0	2	2	4	7	8	8
	0	1	2	3	4	5	6

Очевидно, сега $C[k] = n$. Същината на алгоритъма е в четвъртия **for**-цикъл (редове 8–10). В началото $i = 8$, $A[8] = 4$, $C[4] = 7$. $B[7]$ става 4:

B	3	6	4	1	3	4	4	4
	1	2	3	4	5	6	7	8

а $C[\,]$ става:

C	0	2	2	4	6	8	8
	0	1	2	3	4	5	6

В червено е току-що намаленият елемент (заради ред 10) на $C[\,]$. После $i = 7$, $A[7] = 1$, $C[1] = 2$. $B[2]$ става 1:

B	3	1	4	1	3	4	4	4
	1	2	3	4	5	6	7	8

а $C[\,]$ става:

C	0	1	2	4	6	8	8
	0	1	2	3	4	5	6

После $i = 6$, $A[6] = 4$, $C[4] = 6$. $B[6]$ става 4:

B		1				4	4	
	1	2	3	4	5	6	7	8

а $C[\,]$ става:

C	0	1	2	4	5	8	8
	0	1	2	3	4	5	6

И така нататък. В края на алгоритъма $B[\,]$ е:

B	1	1	3	3	4	4	4	6
	1	2	3	4	5	6	7	8

Сега ща докажем коректността на Counting Sort. Ще въведем следната нотация: ако $Z[1, \dots, m]$ е масив, j е индекс, такъв че $1 \leqslant j \leqslant m$, и x е елемент, който може да бъде елемент на $Z[\,]$, тогава $\#(x, j, Z)$ означава броя на появяванията на x в подмасива $Z[1, \dots, j]$.

Лема 1. *След приключването на втория* ***for****-цикъл (редове 4–5), за* $1 \leqslant j \leqslant k$ *е вярно, че* $C[j] = \#(j, n, A)$.

Доказателство:
Следното твърдение е инварианта на цикъла за втория **for**-цикъл:

> Всеки път, когато изпълнението е на ред 4, за всеки елемент $C[j]$, където $1 \leqslant j \leqslant k$, е изпълнено $C[j] = \#(j, i-1, A)$.

База. При първото достигане на ред 4, всички елементи на $C[\,]$ са нули. От друга страна, i е 1, и така $A[1, \dots, i-1]$ е празен. Твърдението е вярно.

Поддръжка. Да допуснем, че твърдението е в сила при дадено достигане на ред 4, което не е последното. Нека стойността на $C[A[i]]$ е y в момента, в който изпълнението е на ред 5. По индуктивното предположение, $y = \#(A[i], i-1, A)$. Очевидно е, че $\#(A[i], i-1, A) + 1 = \#(A[i], i, A)$. След изпълнението на ред 5, $C[A[i]]$ се увеличава с единица, така че $C[A[i]]$ става равно на $\#(A[i], i, A)$. Тъй като всички останали елементи на $C[\,]$ (освен $C[A[i]]$) остават непроменени от текущото изпълнение на **for**-цикъла, е вярно, че:

- fза всеки елемент $C[j]$ освен $C[A[i]]$, $C[j] = \#(j, i-1, A)$, а също така $C[j] = \#(j, i, A)$
- $C[A[i]] = \#(A[i], i, A)$.

Като цяло, за всеки елемент $C[j]$ е в сила $C[j] = \#(j, i, A)$. Това е преди инкрементирането на i. След инкрементирането на i, $C[j] = \#(j, i-1, A)$ за всяко j, такова че $1 \leqslant j \leqslant k$.

Терминация. Да разгледаме момента, в който изпълнението е на ред 4 за последен път. Очевидно i е $n+1$. Заместваме i с $n+1$ в инвариантата и получаваме “за всеки елемент $C[j]$, където $1 \leqslant j \leqslant k$, е в сила $C[j] = \#(j, n, A)$.” □

Лема 2. *След приключването на третия* ***for****-цикъл (редове 6–7), за* $1 \leqslant j \leqslant k$*, елементът* $C[j]$ *съдържа броя на елементите от* $A[\,]$*, които са* $\leqslant j$.

Доказателство:
Следното твърдение е инварианта на третия **for**-цикъл:

> Всеки път, когато изпълнението е на ред 6, за всяко j, такова че $0 \leqslant j \leqslant i-1$, $C[j] = \sum_{t=1}^{j} \#(t,n,A)$.

База. При първото достигане на ред 6, i е 1, така че твърдението е $C[0] = \sum_{t=1}^{0} \#(t,n,A) = 0$. Но $C[0]$ е наистина 0, защото то се инициализира с 0 от първия **for**-цикъл и вторият **for**-цикъл не му присвоява нищо (понеже $A[i]$ не може да е 0). ✓

Поддръжка. Да допуснем, че твърдението е в сила при някое достигане на ред 6, което не е последното. Елементът $C[i]$ не е променян (засега) от изпълнението на третия **for**-цикъл, така че съгласно Лема 1, $C[i] = \#(i,n,A)$. По индуктивното предположение, $C[i-1] = \sum_{t=1}^{i-1} \#(t,n,A)$. След изпълнението на ред 7 имаме $C[i] = \sum_{t=1}^{i} \#(t,n,A)$. По отношение на новата стойност на i, за всяко j, такова че $0 \leqslant j \leqslant i-1$, $C[j] = \sum_{t=1}^{j} \#(t,n,A)$.

Терминация. Да разгледаме момента, в който изпълнението е на ред 6 за последен път. Очевидно i equals $n+1$. Заместваме i с $n+1$ в инвариантата и получавама "за всяко j, такова че $0 \leqslant j \leqslant n$, $C[j] = \sum_{t=1}^{j} \#(t,n,A)$." □

За всяко j, такова че $1 \leqslant j \leqslant k$, ще наричаме j *съществено*, ако има поне един елемент на $A[\,]$ със стойност j. Съгласно Лема 1, ненулевите елементи на $C[\,]$ след втория **for**-цикъл са точно елементите, чиито индекси са съществени. За всеки x от $A[\,]$ дефинираме понятието *правилното място на* x. Правилното място на x е броя на елементите от $A[\,]$, които са по-малки от x, плюс броя на елементите равни на x, които са вляво от x, плюс едно. С други думи, правилното място на x е индексът му в масива след изпълнението на стабилен сортиращ алгоритъм върху масива.

Лема 3. Counting Sort *е стабилен сортиращ алгоритъм.*

Proof:
Следното твърдение е инварианта на цикъла за четвъртия **for**-цикъл (редове 8–10):

> Всеки път, когато изпълнението е на ред 8, за всяко j, такова че $1 \leqslant j \leqslant k$ и j е съществено, $C[j]$ е правилното място на най-десния елемент от подмасива $A[1,\ldots,i]$, който има стойност j. Нещо повече, всички елементи от $A[i+1,\ldots,n]$ са на своите правилни места в $B[\,]$.

База. При първото изпълнение на ред 6, i е n. Първата част от инвариантата гласи "за всяко j, такова че $1 \leqslant j \leqslant k$ и j е съществено, $C[j]$ е правилното място на най-десния елемент от подмасива $A[1,\ldots,n]$, който има стойност j". Забележете, че $C[\,]$ все още не е променян от четвъртия цикъл, така че Лема 2 е в сила. За всяка стойност j, която се появява в $A[\,]$, правилното място на най-дясното j в $A[\,]$ е равно на сумата от броевете на елементите със стойност $\leqslant j$. Според Лема 2, $C[j]$ е равно точно на тази сума.

Да разгледаме втората част от инвариантата. Подмасивът $A[i+1,\ldots,n] = A[n+1,\ldots,n]$ е празен, така че твърдението е в сила. ✓

Поддръжка. Да допуснем, че твърдението е в сила при някое достигане на ред 8, което не е последното. Стойността на $A[i]$ е някое съществено цяло число j между 1 и k. Нещо повече, то е <u>**най-десният**</u> елемент в $A[1,\ldots,i]$ със стойност j. Съгласно индуктивното предположение, неговото правилно място е $C[A[i]]$ и алгоритъмът го копира точно там (ред 9).

Да допуснем, че има и други елементи със стойност j в $A[1,\ldots,i]$. На ред 10, $C[A[i]]$ бива декрементирано, така че то вече съдържа правилното място на j в $A[1,\ldots,i-1]$. След като i бъде декрементирано е вярно, че елементът на $C[\,]$, който току-що беше декрементиран съдържа правилното място на най-дясното j в $A[1,\ldots,i]$. Тъй като всички останали елементи на $C[\,]$ са непроменени, първата част от инвариантата е в сила.

Сега да допуснем, че в $A[1,\ldots,i]$ няма други елементи със стойност j. Тогава j не е съществен по отношение на останалата част на $A[\,]$ (която предстои да бъде сканирана от четвъртия цикъл), така че стойността на $C[A[i]]$ е без значение за алгоритъма оттук нататък. Наистина, след декрементирането на ред 10, $C[A[i]]$ сочи клетка в B, която е мястото на друг елемент (а не на j). Но, както казахме, стойността на това $C[A[i]]$ няма да бъде използвана до края на алгоритъма, понеже тази стойност на $A[i]$ няма да се среща повече. Първата част от инвариантата се запазва и в този случай.

Сега ще докажем втората част от инвариантата. Да разгледаме момента, когато изпълнението е на ред 8 в началото на текущата итерация. Съгласно индуктивното предположение, всички елементи от $A[i+1,\ldots,n]$ са на своите правилни места в $B[\,]$. Току-що доказахме, че по време на това изпълнение на цикъла, елементът $A[i]$

бива копиран на правилното място. Тогава всички елементи от $A[i, \ldots, n]$ са на своите правилни места в $B[\,]$. По отношение на новата стойност на i, вярно е, че всички елементи от $A[i+1, \ldots, n]$ са на своите правилни места в $B[\,]$.

Терминация. При завършването на цикъла, $i = 0$. Заместваме i с 0 във втората част на инварианта и получаваме "всички елементи от $A[1, \ldots, n]$ са своите правилни места в $B[\,]$." □

Анализът на сложността е тривиален: Counting Sort работи във време $\Theta(n+k)$. Ако $k = O(n)$, както често се случва при използването на този алгоритъм, сложността по време е $\Theta(n)$. Това е значително подобрение спрямо $\Theta(n \lg n)$ на бързите сортиращи алгоритми, базирани на директни сравнения. Сложността по памет е същата: $\Theta(n+k)$, а ако $k = O(n)$, та е $\Theta(n)$. Следователно, алгоритъмът не е *in-place*.

3 Radix Sort

Radix Sort е алгоритъмът, използван за сортиране на перфокарти. Перфокарта, на английски *punched card* или *Hollerith card*, е цифров носител на информация, използван преди появата на работещи компютри, и използван до 70те и дори 80те години на 20 век в компютрите. Работещ компютър, в който информацията се въвежда с перфокарти, в днешно време едво ли може да бъде намерен извън музея, но снимки на перфокарти се намират лесно по Интернет, примерно тази снимка от блога на Elodie. Стандартната Холерит перфокарта има 80 колони, а във всяка колона може да бъде пробита дупка на точно едно измежду дванадесет възможни места. Всяка колона представлява символ. Ако символът е (десетична) цифра, дупката е на една от десетте позиции, маркирани с цифрите. Другите две позиции са за не-цифрова информация. Ако картата записва число, то може да има най-много 80 десетични разряда, колкото са колоните.

Задачата е, при дадено множество карти, които записват числа, картите да се сортират от електро-механично устройство, така че записаните числа да са в ненамаляващ ред. Електро-механичното устройство може да чете в даден момент точно един от десетичните разряди, с други думи, една от колоните, като допира вертикално разположени игли до картата и отчита коя игла потъва (през вече направената дупка), по този начин отчитайки коя е цифрата от съответния разряд.

Първата идея за сортиращ алгоритъм, който да управлява това електро-механично устройство, е да сортираме картите по най-старшия разряд, после по следващия, и така нататък, до най-младшия разряд. Недостатъкът на този подход е, че след сортирането по най-старши разряд трябва да държим картите в десет различни купчини[†] до края, след сортирането по втори най-старши разряд купчините ще станат сто[‡] и така нататък, което не е практично. Правилният подход е разрядите (колоните) да се тълкуват като ключове: старшият разряд е първичният ключ, вторият най-старши е вторичният, и така нататък, най-младшият разряд е 80-ичния ключ. Както знаем от лекцията по сортиране, ако сортираме цялата купчина (без да я разбиваме на подкупчини и подподкупчини и т. н.) първо по 80-ичния ключ, после от 79-ичния ключ, и така нататък, и най-накрая по първичния ключ, ще имаме правилно сортирани карти. Но: **при условие, че сортираме със стабилен сортиращ алгоритъм**. Стабилността гарантира, че когато сортираме по колона i, ако има няколко карти с една и съща цифра в колона i, техният взаимен порядък няма да се наруши; с други думи, вече извършените сортирания по колони $i+1$, $i+2$ и така нататък са определили правилното им взаимно разположение.

Ще дадем пример за работата на Radix Sort. Ще сортираме числата 7444, 1320, 1692, 8185, 4007, 6281, 5139 и 7921. Представяме си, че са написани едно над друго в този ред, и започваме да сортираме от колоната на най-младшите разряди към колоната на най-старшите разряди. Колоната със сив фон отбелязва по кои разряди сортираме в момента.

[†]Купчините биха били десет, ако и десетте цифри се появяват като най-старша цифра на някоя карта.
[‡]Ще станат сто, ако и десетте цифри се появяват като и като най-старша цифра, и като втора най-старша цифра, на някоя карта.

7 4 4 4		1 3 2 0		4 0 0 7		4 0 0 7		1 3 2 0
1 3 2 0		6 2 8 1		1 3 2 0		8 1 8 5		1 6 9 2
1 6 9 2		7 9 2 1		7 9 2 1		5 1 3 9		4 0 0 7
8 1 8 5	→	1 6 9 2	→	5 1 3 9	→	6 2 8 1	→	5 1 3 9
4 0 0 7		7 4 4 4		7 4 4 4		1 3 2 0		6 2 8 1
6 2 8 1		8 1 8 5		6 2 8 1		7 4 4 4		7 4 4 4
5 1 3 9		4 0 0 7		8 1 8 5		1 6 9 2		7 9 2 1
7 9 2 1		5 1 3 9		1 6 9 2		7 9 2 1		8 1 8 5

Псевдокодът на Radix Sort е съвсем прост (съгласно [CLRS09]):

Radix Sort($A[1, \ldots, n]$: положителни числа, записани с един и същи брой цифри; d: броят на цифрите)
1 (∗ най-младшата цифра е номер 1, най-старшата цифра е номер d ∗)
2 **for** $i \leftarrow 1$ **to** d
3   сортирай $A[\,]$ по цифра i със стабилен сортиращ алгоритъм

Очевиден кандидат за стабилен сортиращ алгоритъм е Counting Sort. Коректността на Radix Sort е очевидна, имайки предвид това, което знаем за стабилните сортирания. Ако използваме Counting Sort като стабилен сортиращ алгоритъм, сложността на Radix Sort е $\Theta(d(n+k))$, където k е броят на възможните цифри. Ако k е константа, примерно 10, то сложността по време е $\Theta(dn)$.

Използвайки Radix Sort, можем да решим следната задача: да се сортира в линейно време масив от n числа, всяко от което принадлежи на множеството $\{1, 2, \ldots, n^2\}$. Ако опитаме директно да я решим с Counting Sort, решението ще работи в $\Theta(n^2)$, защото работният масив на Counting Sort ще бъде с размер n^2. Но ако разгледаме записите на тези числа, примерно в двоична позиционна бройна система, ще видим, че всяко от тях се записва с $\lfloor \lg n^2 \rfloor + 1 = \lfloor 2 \lg n^2 \rfloor + 1 \approx 2 \lg n$ бита. Ако разделим всеки бинарен запис на две части, всяка с по $\approx \lg n$ бита, може да смятаме дясната част за младши ключ, а лявата, за старши ключ. Прилагайки Radix Sort (който на свой ред ползва Counting Sort) първо по младшите ключове, а после по старшите, имаме решение във време $\Theta(2n) = \Theta(n)$. Това, което ни помогна да конструираме линеен алгоритъм е, че прилагаме Counting Sort два пъти, но при всяко прилагане, работният масив е линеен, защото и младшите, и старшите ключове са с големина $\approx \lg n$, което означава, че могат да има най-много $\Theta(n)$ различни стойности.

Литература

[CLRS09] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms, Third Edition*. The MIT Press, 3rd edition, 2009.