Търсене на най-малък елемент в подмасив и на най-близък общ предшественик в кореново дърво

Търсене на най-малък елемент в подмасив

Даден е масив $A[1 \ldots n]$ с нареден базов тип. Трябва да му извършим предварителна обработка, след което да отговорим на множество заявки $(x\ ;\ y)$ за индекса на най-малък елемент на подмасива $A[x \ldots y]$.

За целта строим таблица M с n реда и $k + 1$ стълба, където $k = \lfloor \log_2 n \rfloor$. В клетката $M[i][j]$ от таблицата ще пазим индекса на най-малкия елемент в подмасива $A[i \ldots i + 2^j - 1]$ с дължина 2^j, където $1 \le i \le n$ и $0 \le j \le k$. Таблицата не е правоъгълна, тъй като $i + 2^j - 1 \le n$. Построяваме я така:

```
M[1...n][0...k] ← Предварителна_обработка(A[1...n])
 1) for i ← 1 to n do
 2)    M[i][0] ← i
 3) for j ← 1 to k do
 4)    for i ← 1 to n - 2^j + 1 do
 5)       p ← M[i][j - 1]
 6)       r ← M[i + 2^(j-1)][j - 1]
 7)       if A[p] < A[r]
 8)          M[i][j] ← p
 9)       else
10)          M[i][j] ← r
```

Построяването на таблицата е предварителната обработка на входните данни и отнема време $\Theta(n \log n)$. Таблицата изразходва памет $\Theta(n \log n)$.

Пример как може да се използва таблицата за обработка на заявката $(2\ ;\ 6)$:

$$\overbrace{M[3][2] = r, \quad A[r] = \min A[3 \ldots 6]}$$

$A[1]$	$A[2]$	$A[3]$	$A[4]$	$A[5]$	$A[6]$	$A[7]$	$A[8]$

$$\overbrace{M[2][2] = p, \quad A[p] = \min A[2 \ldots 5]}$$

$$\overbrace{\min A[2 \ldots 6] = \min \{ A[p]\ ;\ A[r] \}}$$

В общия случай всяка заявка (x ; y) се обработва така:

```
Най-малък_елемент(A, M, x, y)
1) j ← ⌊log2(y - x + 1)⌋
2) p ← M[x][j]
3) r ← M[y - 2^j + 1][j]
4) if A[p] < A[r]
5)     return p
6) else
7)     return r
```

Една заявка изисква памет $\Theta(1)$ и време $\Theta(\log(y - x)) = O(\log n)$, като асимптотичната горна граница е точна: достига се например за $x = 1$, $y = n$. Това време се изразходва за пресмятането на логаритъма и може да се намали до $\Theta(1)$ чрез предварително логаритмуване на числата от 1 до n включително по формулата $\lfloor \log_2 s \rfloor = 1 + \lfloor \log_2 (s/2) \rfloor$ с последователно нарастване на s (това става за време $\Theta(n)$ и не забавя предварителната обработка съществено).

```
log2[1...n] ← Логаритми(n)
1) log2[1] ← 0
2) for s ← 2 to n do
3)     log2[s] ← 1 + log2[ ⌊s/2⌋ ]
```

Делението на последния ред е целочислено и се извършва най-бързо с помощта на побитова операция: битовете на `s` се отместват надясно с една позиция. Степените на двойката от двете предишни процедури се намират с отместване на битовете наляво (операцията е вградена в процесора и отнема време $\Theta(1)$).

Пример:

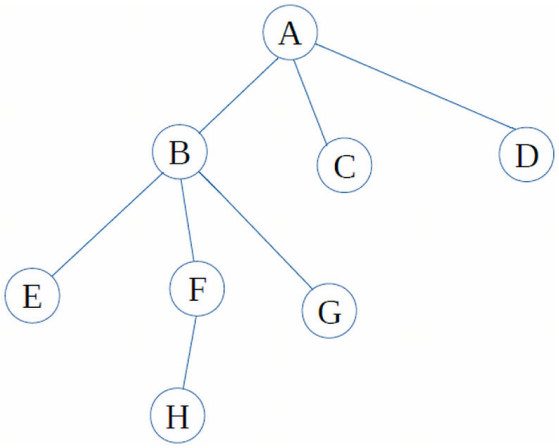
число	1	2	3	4	5	6	7	8
логаритъм	0	1	1	2	2	2	2	3

Помощният числов масив log2 изразходва памет $\Theta(n)$ за съхранението си и време $\Theta(n)$ за пресмятането си. Таблицата M отнема време и памет $\Theta(n \log n)$, което е асимптотично повече от $\Theta(n)$. В този смисъл масивът от логаритми не утежнява реализацията съществено. Напротив, този допълнителен масив ускорява обработката на заявки, намалявайки времето от $\Theta(\log n)$ на $\Theta(1)$.

Търсене на най-близък общ предшественик в кореново дърво

Най-близкият общ предшественик на два върха на кореново дърво е този от общите им предшественици, който е разположен на най-голяма дълбочина (най-далече от корена). Най-близък общ предшественик винаги съществува, защото коренът е предшественик на всеки връх, включително на себе си.

Пример: Върхът B е най-близкият общ предшественик на върховете E и H на дървото, показано на чертежа.



Отново се иска да отговорим на множество заявки за двойки върхове на едно и също кореново дърво T. Имаме право да обработим T предварително. Предполагаме, че дървото T е представено със списъци на преките наследници.

Означаваме с n броя на върховете на T.

Първи алгоритъм: Предварителната обработка се състои в намирането на дълбочините на всички върхове. Това става с едно обхождане в ширина на дървото T за време $\Theta(n)$. Всеки връх запазва указател към родителя си.

След това всяка постъпила заявка се обработва така:

1) Върха, който е на по-голяма дълбочина, заменяме с неговия родител. Продължаваме до изравняване на височините на двата върха.
2) Ако двата върха са различни, заменяме всеки от тях с неговия родител. Продължаваме, докато двата върха съвпаднат.
3) Общата им стойност е търсеният най-близък общ предшественик.

Пример: Ако търсим най-близък общ предшественик на върховете E и H на показаното дърво, то стъпка 1 заменя H с F, стъпка 2 заменя E и F с B, а стъпка 3 обявява върха B за най-близък общ предшественик на E и H.

Отделната заявка се обработва за време $\Theta(h)$ при най-лоши входни данни, където h е височината на дървото. Очевидно $\Theta(h) = O(n)$, като горната граница се достига, когато дървото се изроди в път с начало корена. В такъв случай обработката на отделна заявка изисква време $\Theta(n)$, тоест тази обработка е твърде бавна.

Втори алгоритъм: Предварителната обработка се състои в обхождане на дървото T в дълбочина. Подреждаме върховете в масив, всеки връх се среща толкова пъти, колкото пъти преминаваме през него — при първото достигане и при всяко връщане от пряк наследник. Масивът съдържа $2n - 1$ елемента, където n е броят на върховете на дървото. При обхождането пресмятаме и дълбочините им. Тази процедура изразходва време и памет $\Theta(n)$.

Пример: За дървото, показано по-горе, масивът ще изглежда така:

индекс	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
връх	*A*	*B*	*E*	*B*	*F*	*H*	*F*	*B*	*G*	*B*	*A*	*C*	*A*	*D*	*A*
дълбочина	0	1	2	1	2	3	2	1	2	1	0	1	0	1	0

Всеки връх на дървото се среща в този масив поне веднъж. За всеки връх пазим една от позициите, на които се среща. Няма значение коя точно пазим, можем да изберем например първата:

връх	*A*	*B*	*C*	*D*	*E*	*F*	*G*	*H*
индекс	1	2	12	14	3	5	9	6

Това се прави в процеса на обхождане на дървото и изисква допълнително време и памет $\Theta(n)$.

Обработката на отделна заявка се извършва така: по дадени два върха намираме индексите им x и y в първия масив с помощта на втория, след което намираме най-малката дълбочина в подмасива на първия масив с индекси между x и y включително. Върхът, стоящ на същата позиция, представлява най-близкият общ предшественик на дадените два върха.

Пример: За върховете E и H чрез втория масив намираме индекси 3 и 6 съответно. Подмасивът на първия масив с индекси от 3 до 6 вкл. притежава минимална дълбочина 1, която се намира на позиция № 4. На същата позиция стои върхът B. Тъкмо той е най-близкият общ предшественик на E и H.

По този начин търсенето на най-близкия общ предшественик се свежда до търсенето на най-малък елемент на подмасив. Както знаем, тази задача може да се реши, като на първия масив се направи допълнителна обработка, която се състои в построяването на таблицата M и изисква време и памет $\Theta(n \log n)$.

Така в крайна сметка предварителната обработка на дървото притежава сложност по време и памет $\Theta(n \log n)$. Затова пък после обработката на заявка придобива сложност по време и памет $\Theta(1)$.