DAA Упражнение 7 (Дизайн на алгоритми с масиви)

Във всички задачи се търси възможно най-бързо решение (в асимптотичен смисъл) по време и по памет.

зад.1/ Даден е масив от цели числа с големина ≥2. Да се намерят индексите на тези два елемента, такива че произведението им е максимално.

Решение: Ако масивът съдържаше само положителни числа, то тогава трябва да намерим индексите на най-големият елемент и на вторият по големина. Важното съображение, което трябва да направим тук е, че произведението на двата най-малки елемента може да е по-голямо от произведението на двата най-големи елемента.

```
SOLUTION(A[1...n]: масив от цели числа)
1. S_1 ← 1, S_2 ← 1, b_1 ← 1, b_2 ← 1   // S_1 - индексът на най-малкият ел.
                                        // b_1 - индексът на най-големият ел.
                                        // S_2 - индексът на 2рият най-малък ел.
                                        // b_2 - индексът на 2рият най-голям ел.
                                        // в подмасива A[1..i-1]
2. for i ← 1 to n
3.     if A[i] > A[b_1]
4.         b_2 ← b_1
5.         b_1 ← i
6.     else if A[i] > A[b_2]
7.         b_2 ← i
8.     if A[i] < A[S_1]
9.         S_2 ← S_1
10.        S_1 ← i
11.    else if A[i] < A[S_2]
12.        S_2 ← i
13. if A[S_1] * A[S_2] > A[b_1] * A[b_2]
14.     return (S_1, S_2)
15. return (b_1, b_2)
```

Разглеждаме трите различни случая за i-тият елемент. 1) Той да е НГ елемент. 2) Той да е 2рият НГ елемент. 3) Той да е 3ти по големина или по-малък.

Правим аналогичното и за най-малък елемент.

зад.2/ Даден е масив с елементи от $\mathbb{N} \setminus \{0\}$. Да се намери най-малкото естествено число, което не се среща в масива.

Решение: Можем да сортираме масива за $\Theta(n \lg n)$ време и след това линейно да обходим веднъж като върнем първият индекс i за който A[i] != i (ако такъв не съществува, то търсеният отговор е n+1, понеже масива съдържа 1,2,...,n). Това решение обаче е прекалено бавно.

Друга идея която може да опитаме е да направим counting sort, като в помощния масив ще маркираме колко пъти се срещат числата от 1 до n. След това с едно линейно обхождане ще върнем елементът, който се среща 0 пъти (или n+1 ако всички елементи 1,..,n се срещат точно по веднъж). Това решение е оптимално по време, но не и по памет. Ако не ни е казано, че не можем да модифицираме оригиналния масив, то може да маркираме срещането на елемент i, като по някакъв начин маркираме елемента на индекс i (например чрез умножение по -1, или добавяне на $\frac{1}{2n}$ към елемента). Накрая първото индекса, който не е маркиран ще е естественото число, което не се среща (или n+1, ако всички индекси са маркирани). Псевдокоде реализиращ тази идея:

```
SOLUTION(A[1..n]: масив с елементи от ℕ⁺)
1. for i ← 1 to n
2.     A[A[i]] ← min(A[A[i]], -A[A[i]]) // за предотвратим двойно маркиране,
                                        // т.е. "отмаркиране"
3. for i ← 1 to n
4.     if A[i] > 0 // ако елемента не е бил маркиран
5.         return i // връщаме индекса, а не самият елемент!
6. return n+1 // елементите на масива са някоя перпутация на 1,..,n
```

зад. 3/ Даден е масив от цели числа. Предложете двойка алгоритми индекс и заявка, които отговарят на въпроса "Каква е сумата на елементите от индекс i до индекс j?" (i<j за да си спестим една проверка и i,j са валидни индекси)

Заб: Идеята на задачи от този вид е да се изпълни един път алгоритъма за предварителна обработка (индекса), който прави заявките максимално бързи, понеже те ще са много на брой.

Решение: Един възможен начин да направим заявките за константно време е ако предварително изчислим колко е $\sum_{c=i}^{j} A[c]$ за всяко i<j. Проблемът с този подход е че предварителната обработка ще е прекалено тромава и ще изразходва прекалено много допълнителна памет (от порядъка на n^2). Може за линейно време да направим така че в клетка A[i] от първоначалния масив да стои $\sum_{c=1}^{i} A[c]$. Тогава ако искаме $\sum_{c=i}^{j} A[c]$ това ще е същото като $\sum_{c=1}^{j} A[c] - \sum_{c=1}^{i-1} A[c]$, което изчисление може да стане за константно време и само така ползваме само константна доп. памет.

Следният псевдокод реализира идеята:

```
INDEX(A[1...n]: масив от цели числа)
1. A[0] ← 0
2. for i ← 1 to n
3.     A[i] ← A[i] + A[i-1]
4. return A[0...n]
```

```
QUERY(i,j: индекси в масива)
1. return A[j] - A[i-1]
```

зад.4/ Дадена е редица от n на брой елемента, като първоначално има един "лош" елемент на някоя от позициите. Имаме право да задаваме въпроса "На i-ти индекс ли се намира "лошият" елемент в момента?" на което ни се отговаря с Да/Не. След всеки наш въпрос, "лошият" елемент се премества с една стъпка наляво или надясно в редицата, като със сигурност не излиза извън рамките на самата редица. Предложете алгоритъм, който със сигурност открива "лошия" елемент.

Решение: Имаме 4 случая на задачата. Индексът, където първоначално се намира "лошият" елемент да е четен/нечетен и дължината на редицата да е четна/нечетна. Ако първоначално "лошият" елемент е на нечетен индекс, то с едно линейно обхождане запознаваши от 1ви елемент, след това 2ри и така до n-ти със сигурност ще го намерим. Това е така защото всеки път питаме за четност на която се намира той и няма как да "избяга" отляво на вече питан елемент. Понеже ~~как~~ след всеки зададен въпрос елементът сменя четността си, то ако редицата е ~~първоначално~~ с нечетен брой елементи и първоначалната позиция на "лошия" елемент е на четна позиция, то след първото линейно обхождане "лошият" елемент вече е на нечетна позиция и преминаваме към първоначалните 2 случая. Последният случай е ако редицата е с четна дължина и първоначално "лошият" елемент е на четен индекс. Тогава след първото обхождане той няма да смени четността си, но ние можем да я сменим, като повторно питаме за първият елемент. Следната идея е реализирана със следния псевдокод:

SOLUTION(A[1..n] - редица от числа)
1. for i ← 1 to n
2. if (На i-ти индекс ли се намира "лошият" елемент? = Да)
3. return "Намерихме елемента."
4. if (n%2 = 0) //n е четно
5. if (На 1-ви индекс ли се намира "лошият" елемент? = Да)
6. return "Намерихме елемента."
7. for i ← 1 to n
8. if(.... = Да)
9. return "Намерихме елемента."

зад.5/ Даден е масив от числа. и число k. Да се намери k-тият по големина елемент.

Решение: Едно решение е да сортираме масива и да върнем k-тият елемент. Това решение е със сложност $\Theta(n \lg n)$, но има по-бързо решение в асимптотичния смисъл. Това е да използваме алгоритъма SELECT, който е изучаван на лекции и работи в линейно време.

Заб: Ако на практика искате да решите задачата, то първото решение би било в пъти по-бързо. Това е от части заради мултипликативната константа на SELECT, която е около 5-10 и това как работят различните нива на памет в процесора.

зад.6/ Даден ни е масив, който съдържа числата $0, \ldots, n$ без повторения, като едно от тях липсва. Да се намери липсващото число.

Решение: Знаем че $1+2+\ldots+n = \frac{n(n+1)}{2}$. Можем да сумираме всички числа в масива и да я извадим от $\frac{n(n+1)}{2}$ и така ще намерим липсващият елемент.

зад.7/ Даден е масив с цели числа. Предложете алгоритъм, който преномерира елементите в масива така че всички отрицателни елементи са отляво на положителните.

Решение: Едно решение е да сортираме елементите. Тогава всички отрицателни елементи ще са отляво на положителните. Проблемът е че алгоритъма е със сложност $\Theta(n \lg n)$. Можем да постигнем същото, ако използваме алгоритъма Partition и подадем за pivot 0. Тогава за линейно време ще получим искането.