**Анализ на решението на задача**

**Мярка**

Първата подзадача е за 10 точки. Тя е предвидена за наивното решение. Нямаме заявки, така че ще се концентрираме само за определяне на мярката на входния масив. Последователно намираме търсените минимуми за подинтервалите на масива. Нека сме фиксирали десен край *r*. Тогава последователно местим левия край от *r* до началото. Едновременно с това поддържаме текущия минимум за интервала и го натрупваме в сумата, която ще е крайната мярка на входния масив. Сложността е .

Втората подзадача е за 30 точки. Трябва да оптимизираме намирането на тази сума. Има различни подходи, като тук ще говорим за малко по-бавни подходи от тези, предвидени за трета подзадача. Може би най-интуитивен е следният подход, тип разделяй и владей. Нека разглеждаме някой минимум на целия интервал. Ще означим неговия индекс с *ind*. Тогава е ясно, че той ще участва в сумата за всички интервали, които го съдържат. Лесно можем да сметнем тяхната бройка – за ляв край имаме (*ind*+1) възможности (включително 0 и *ind*), а за десния край имаме (***N***-*ind*+1) възможности, независими от другите. Това означава, че интервалите, които го съдържат са (*ind*+1)\*(***N***-*ind*+1) на брой, т.е. към сумата трябва да прибавим (*ind*+1)\*(***N***-*ind*+1)\*a[*ind*]. С това сме обработили всички интервали, които минават през *ind*. Това означава, че можем независимо да се концентрираме върху интервали изцяло наляво от *ind* или изцяло надясно от *ind*. При тях можем да приложим същите разсъждения. Така рекурсивно ще изчерпим всички интервали. Ясно е, че тази рекурсивна процедура ще има ***N*** стъпки, защото всеки път разглеждаме някой конкретен индекс и по надолу в рекурсията не го гледаме повече. Търсенето на минимум вече е задачата *RMQ*, за която е загатнато в условието. Авторът е използвал *sparse table* за намиране на минимумите, т.е. ***N***\**log(****N****)* предпроцесване, но константна заявка. Интересен факт е, че тази описана рекурсивна процедура строи така нареченото *Декартово дърво*. По принцип то може да се строи и с линейна сложност, като този алгоритъм донякъде се припокрива с това, за което ще говорим в трета подзадача. Очаквана сложност за решение на тази подзадача е .

Третата подзадача е за 20 точки. Вече ни трябва линейно решение за намиране на мярката. Този път ще смятаме мярката с натрупване, т.е. ще смятаме за интервалите, завършващи в a[0], после тези, завършващи в a[1], … накрая интервалите завършващи в a[**N**-1]. Нека сме натрупали мярката до a[k] и сега искаме да намерим минимумите за интервалите [0, k+1]; [1, k+1]; …; [k+1; k+1]. Полезно би било да знаем колко наляво числото a[k+1] ще е минимум. Нека това е изпълнено до индекс ind. Това означава, че за всички интервали с ляв край ≥ ind минимума ще е a[k+1], т.е. към мярката трябва да натрупаме ((k+1)-ind+1)\*a[k+1]. Друго важно нещо е, че всъщност интервалите с ляв край < ind, ще имат за минимум стари стойности. Това ни навежда на мисълта, че можем да следим кои числа до къде са минимум със стандартната техника със стек. Така на всяка итерация от него, махаме елементи, докато намерим по-малък елемент от текущия, защото така намираме именно и тази позиция ind. За да можем да сметнем останалата сума (за левите краища < ind), можем например да поддържаме текуща сума на минимумите за интервалите с ляв край 0, 1, 2, …, k+1 и десен край последната позиция k+1. Когато добавим a[k+1], тази текуща сума се увеличава с ((k+1)-ind+1)\*a[k+1], за да отчетем че вече минимумът за интервалите с ляв край ind, ind+1, …, k+1 и десен край k+1 вече е a[k+1]. Съответно, когато махахме елементи, трябваше подходящо да намаляме тази текуща сума (с което да отчетем, че вече ще имаме нов минимум за интервалите с такива леви краища). Описаният алгоритъм за всеки индекс работи с амортизирана константа, защото общо всеки елемент влиза веднъж и излиза веднъж от стека. Очакваната сложност за решение на тази подзадача е .

Четвъртата подзадача е за 40 точки. Тук вече трябва да работим и със заявките. Важно е, че можем да работим с тях офлайн. Освен това те са независими една от друга. Така че може за всяка позиция във вектор да запомним евентуалните смени на стойността. Горният алгоритъм лесно се модифицира, така че да намира мярката на префиксните подмасиви за заявките. Лесно може да се забележи, че минимумите в стека са в намаляващ ред (гледайки от върха към дъното). Така че вместо за всяка стойност да махаме елементи, можем да правим двоично търсене, за да видим за всяка стойност кой е ключовият индекс ind. Тук може би ще е по-удобно да не пазим текуща сума на стека, а по-скоро за всеки минимум, да пазим сумата от него до дъното за съответните леви краища. Така, намирайки с двоично търсене подходящия индекс, този елемент в стека ни дава достатъчна информация да сметнем ако дадения елемент беше със съответната стойност, то какво трябваше да прибавим към мярката. Важно е да се отбележи, че понеже търсим мярката само на префикса, то намирайки за дадена заявка за индекс *i* сумата на минимумите за интервалите, завършващи в *i*, използвайки досега натрупаната мярка получаваме и отговора на съответната заявка. Това е така, понеже не се интересуваме за съответната заявка от числата вдясно. Освен това стека трябва да го променяме, използвайки само оригиналните числа. Крайната сложност на решението е .

*Автор: Илиян Йорданов*