**Анализ на решението на задача  
Пътуване**

Разполагаме с времената за изминаване на поредните метри от пътя в масив с *n* елемента. Търсим *m* последователни елемента в масива, чиято сума е минимална. Един възможен подход е да сумираме с цикъл първите *m* елемента, започвайки от нулевия елемент на масива и това за момента да е рекордно най-краткото време за изминаване на участък. След това последователно пресмятаме в цикъл времето за изминаване на участък, започващ от първия елемент, после за участък, започващ от втория елемент и т.н. Всяка поредна сума от *m* съседни елемента сравняваме с рекорно най-краткото време и ако е по-малка – коригираме рекорда и помним къде е възникнал (индекса на елемент от масива, който е начален за участъка). Една примерна реализация:

**short** t[10000000];

**long long** min\_sum=10000000001,sum;

**int** i,j,n,m,pos\_min;

/// тук са пропуснати някои от действията, напр. четенето от входа

**for** ( i = 0; i < n - m ; i++) {

**for** (sum = j = 0; j < m ; j++) sum += t[i + j];

**if** (sum < min\_sum) min\_sum=sum, pos\_min=i;

}

Макар и очевиден, този начин не е много ефективен, защото изисква огромен брой събирания при големи стойност на *n* и . Да разгледаме, например, колко събирания са необходими за решаване на задачата при *n* = 107 и *m* = 5.106. За определяне на сумата във вътрешния цикъл се извършват *m* добавяния (събиране) на елемент от масива в суматора *sum*. Външният цикъл се изпълнява *n*-*m*+1 пъти или общият брой събирания е  
(*n*-*m*+1)\*m = 25 000 005 000 000. Ясно е, че такъв брой събирания не могат да се извършат за разумно време в състезателни условия, което налага да търсим по-ефективен алгоритъм, при който броят събирания е по-малък.

Да разгледаме какво се случва ако трябва да сумираме *m* елемента от масива, започвайки от елемент *i* и от елемент *i*+1:

sum*i* = *ti* + (*ti*+1 + *ti*+2 + … + *ti*+*m*-1) = *ti* + S (1)

sum*i+*1 = (*ti*+1 + *ti*+2 + … + *ti*+*m*-1) + *ti+m* = S + *ti+m* (2)

От зависимост (1) следва, че:

S = sum*i* – *ti* (3)

Заместваме S в (2) и получаваме:

sum*i+*1 = sum*i* – *ti* + *ti+m* (4)

Т.е. ако знаем sum*i* , определянето на sum*i+*1 става само с две събирания, без значение колко е m !!! От тук ще получим огромна икономия в броя събирания при решаване на задачата. За първия участък ще извършим *m* събирания, тук няма как да икономисаме действия, но за всички останали *n* – *m* участъци ще извършваме само по 2 събирания и така общият брой събирания става *m* + 2\**n* – 2\**m* = 2\**n* – *m*. При *n* = 107 и *m* = 5.106 общия брой събирания се получава 15 000 000. Сравнете го с броя при предишния разглеждан подход по-горе!

В приведения по-долу примерен код са направени незначителни отклонения от описания алгоритъм, главно заради технологични удобства при работа. Например: като индикатор за местоположението на всеки участък от *m* съседни елемента в масива се използва индекса на последния елемент в участъка, вместо на първия.

Сорс-кодът на програмата

#include <stdio.h>

**short** A[10000000];

**long long** min\_sum,sum;

**int** i,n,m,pos\_min;

**int** main() {

scanf(" %d %d",&n, &m);

**while** ( i<m ) {

scanf(" %hd",A+i);

sum += A[i];

i++;

}

min\_sum=sum;

pos\_min=m-1;

**while** ( i<n ) {

scanf(" %hd",A+i);

sum += A[i] - A[i-m];

**if** (sum <= min\_sum) {

min\_sum=sum;

pos\_min=i;

}

i++;

}

printf("%d\n",pos\_min-m+1);

**return** 0;

}

*Автор: Евгений Василев*