СЪСТАВЯНЕ НА АЛГОРИТМИ
ДОМАШНА РАБОТА ПО УЧЕБНАТА ДИСЦИПЛИНА
"ДИЗАЙН И АНАЛИЗ НА АЛГОРИТМИ"
(СУ, ФМИ, ЛЕТЕН СЕМЕСТЪР НА 2019 / 2020 УЧ. Г.)

Задача 1. Имаме шоколадово блокче $2 \times n$. Двама играчи се редуват. Всеки играч избира едно неизядено квадратче и отхапва всички квадратчета нагоре и надясно от избраното (включително избраното квадратче). Квадратчето в долния ляв ъгъл на шоколада е отровно: който бъде принуден да изяде това квадратче, губи играта.

а) Кой от двамата играчи има печеливша стратегия и каква е тя? **(1 точка)**
Отговорът да се обоснове (може и неформално). **(1 точка)**

б) Докажете формално (с полуинвариант), че играта ще завърши. **(1 точка)**

в) Докажете формално (с два инварианта), че стратегията, посочена като отговор на подточка "а", наистина е печеливша. **(2 точки)**

Задача 2. Даден е масив $A[1...n]$ от положителни числа — височини на сгради, разположени последователно (на една линия). Целта ни е да бутнем сградите. Можем да сложим взрив или в началото на редицата, или в края (отляво на първата сграда или отдясно на последната). Всяка сграда пада в посоката, в която е била бутната: ако взривът е бил разположен отляво на сградата, тя пада надясно (и обратно). Взривът сам по себе си събаря само една сграда: ако е сложен в началото на редицата — най-лявата сграда; ако е в края — най-дясната. Обаче всяка сграда при своето падане може да успее или да не успее да събори следващата. По-точно, падащото здание събаря следващата сграда само ако то е по-високо от нея.

Пример: Нека височините на сградите са

20, 10, 8, 9, 15, 15, 17, 14, 15, 16, 17 метра.

Ако сложим взрив в началото на редицата, ще бутнем първата сграда, която при падането си ще събори втората ($20 > 10$), която на свой ред ще събори третата ($10 > 8$); ще остане редицата 9, 15, 15, 17, 14, 15, 16, 17. Ако разполагаме взривовете само в началото, ще ни трябват още четири взрива за сградите с височина 9, 15, 15 и 17 метра, тъй като $9 < 15 = 15 < 17$. Последната от тях ще бутне следващата ($17 > 14$). За останалите три сгради ще ни трябват още три взрива ($14 < 15 < 16 < 17$). Така събаряме сградите с осем взрива.

Можем обаче да постъпим по друг начин. Първия взрив слагаме в левия край и бутаме първите три сгради ($20 > 10 > 8 < 9$). Втория взрив слагаме в края на редицата и събаряме последните четири сгради ($17 > 16 > 15 > 14 < 17$). Остава редицата 9, 15, 15, 17 метра. Разполагаме трети взрив вдясно; той събаря последните две сгради ($17 > 15 = 15$), при което остава редицата 9, 15. За събарянето на тези две сгради е достатъчен един взрив, сложен в края на редицата ($15 > 9$). Оказа се, че четири взрива са достатъчни, няма нужда от осем.

Да се състави алгоритъм, който намира най-малкия брой взривове, нужни за събаряне на сградите, мястото на всеки взрив (в началото или в края на оставащата редица от сгради) и реда на задействане на взривовете. Алгоритъмът да се опише на програмния език Си и да използва само стандартния вход-изход, примитивни числови типове (например float) и масиви. Да не се използват готови функции (библиотеки) освен за стандартния вход-изход.

Алгоритъмът се оценява в зависимост от сложността си в най-лошия случай:

— Ако има сложност по време $T(n) = O(n)$ и памет $M(n) = O(n)$, той носи **3 точки.**

— Ако има сложност по време $T(n) = O(n)$ и памет $M(n) = O(1)$, той носи **5 точки.**

— Ако поне една от сложностите не е $O(n)$, алгоритъмът не носи точки.

Грешни алгоритми не носят точки.

РЕШЕНИЯ

Задача 1 е частен случай на играта “Щрак” (има я в книгата “Математически игри” от Владимир Чуканов). Доказано е, че първият играч има печеливша стратегия, обаче това е чисто твърдение за съществуване: от доказателството не може да се изведе стратегията. Тя е известна само в някои частни случаи — $1 \times n$, $2 \times n$ и $n \times n$. Първият случай е тривиален, третият е симетричен, така че само вторият представлява интерес. Именно вторият случай се разглежда в тази задача.

При играло табло $2 \times n$ първият играч ще спечели, ако на своя първи ход отхапе единичното квадратче в горния десен ъгъл на шоколадовото блокче. Така горният ред става с едно квадратче по-къс от долния ред. Вторият играч ще наруши това съотношение: ако отхапе от горния ред, разликата между двата реда ще стане повече от едно квадратче; ако отхапе от долния ред, двата реда ще се изравнят по дължина. Със следващия си ход първият играч възстановява съотношението: в първия случай отхапва от долния ред толкова квадратчета, колкото вторият играч току-що е отхапал от горния ред; във втория случай отхапва едно квадратче от горния ред — най-дясното. И в двата случая горният ред става с едно квадратче по-къс от долния ред.

По-формално, нека a и b са съответно броят на квадратчетата в горния и в долния ред. Тогава са в сила следните два инварианта:

Инвариант 1: След всеки ход на първия играч е в сила равенството $b - a = 1$.

Инвариант 2: След всеки ход на втория играч е в сила равенството $b - a \neq 1$.

Доказателство: с обикновена индукция по броя на ходовете. Индукцията е взаимна — всеки инвариант следва от другия.

База: В началото на играта $b = a = n$. След първия ход на първия играч (отхапването на единичното квадратче в горния десен ъгъл) имаме равенствата $a = n - 1$, $b = n$, затова $b - a = n - (n - 1) = 1$, тоест в сила е инвариант 1.

Индуктивна стъпка: Предполагаме, че инвариант 1 е в сила след някой непоследен ход на първия играч. Ще докажем, че инвариант 2 важи след непосредствено следващия ход на втория играч. От инвариант 1 следва, че след току-що изиграния ход на първия играч горният ред на шоколадовото блокче съдържа a квадратчета, а долният ред съдържа с едно квадратче повече: $b = a + 1$. По-нататък разглеждаме два случая за хода на втория играч.

Първи случай: Вторият играч отхапва най-десните k квадратчета от долния ред, $1 \leq k \leq b$. От правилата на играта следва, че той е длъжен да отхапе и най-десните $k - 1$ квадратчета от горния ред. Тогава след неговия ход е в сила равенството $a = b$, откъдето $b - a = 0 \neq 1$, тоест важи инвариант 2. Според описаната стратегия първият играч на следващия си ход (ако има такъв) отхапва единичното квадратче в горния десен ъгъл, след което $a = b - 1$, тоест $b - a = 1$, значи е в сила инвариант 1.

Втори случай: Вторият играч отхапва най-десните k квадратчета от горния ред, $1 \leq k \leq a$. След неговия ход броят a намалява с k, а броят b остава непроменен, затова разликата $b - a$ се увеличава с k. Тъй като според индуктивното предположение (инвариант 1) тази разлика е била равна на 1 преди хода на втория играч, то след неговия ход тя ще бъде равна на $k + 1$, тоест $b - a = k + 1 \neq 1$, защото $k > 0$. И така, в сила е инвариант 2: важи равенството $b - a \neq 1$. Първият играч (ако има ход) отговаря, като отхапва същия брой квадратчета от долния ред, тоест отхапва най-десните k квадратчета, при което умаляемото b намалява с k. Ето защо и разликата $b - a$ намалява с k, тоест става равна на $(k + 1) - k = 1$. И така, след поредния ход на първия играч важи равенството $b - a = 1$, тоест в сила е инвариант 1.

Полуинвариант е броят на оставащите квадратчета. Този брой намалява поне с единица на всеки ход, следователно играта продължава най-много $2n$ хода.

Играта завършва, когато бъде изядено цялото шоколадово блокче, тоест при $a = b = 0$. Тогава $b - a = 0 \neq 1$. От инвариант 1 следва, че това не се случва след ход на първия играч. Тогава последният ход е на втория играч, т.е. той изяжда отровното квадратче и губи играта. Следователно описаната стратегия е печеливша за първия играч.

Задача 2. Да наречем един взрив ляв или десен според това, в кой край на редицата от сгради е разположен. Аналогично, всяка сграда е съборена или отляво, или отдясно (било от взрив, било от друга падаща сграда). Ако една сграда е била бутната отляво от падаща сграда, то самата падаща сграда е била съборена отляво. Ако някоя сграда е бутната отляво от взрив, то за да бъде активиран самият взрив, е трябвало първо да бъдат съборени сградите, които са се намирали наляво от взрива преди неговото активиране, а това може да стане само ако те са били бутнати отляво.

Накратко, сградите наляво от сграда, съборена отляво, също са били съборени отляво; а сгради надясно от сграда, съборена отдясно, също са били съборени отдясно. Аналогично, взривовете наляво от ляв взрив са леви, а взривовете надясно от десен взрив са десни.

Следователно съществува разделител, който дели взривовете и сградите на леви и десни. Разделителят представлява място между две последователни сгради или мястото отляво на първата сграда (ако всички взривове са десни), или мястото отдясно на последната сграда (ако всички взривове са леви). За да не работим с дроби, ще кодираме разделителя с индекса на сградата отляво на разделителя. Тоест разделител k означава, че сградите с номера $1, 2, \ldots, k$ са бутнати отляво, а сградите с номера $k+1, k+2, \ldots, n$ — отдясно. Разделител n значи, че всички здания са бутнати отляво, а 0 — че всички са бутнати отдясно.

Достатъчно е да направим алгоритъм, който връща две цели неотрицателни числа — мястото на разделителя и най-малкия брой взривове. Редът на активиране на взривовете се определя от мястото на разделителя: левите взривове се активират отляво надясно, а десните — отдясно наляво. Редът на активиране на левите спрямо десните взривове се определя от съотношението на височините на сградите непосредствено до разделителя: ако по-ниската сграда е отдясно, активираме първо десните взривове (и обратно); ако пък двете сгради са еднакво високи, изборът е произволен: можем да активираме първо левите или първо десните взривове, а можем да ги редуваме.

Възможните стойности на разделителя са $0, 1, 2, \ldots, n$; броят им е $n + 1 = \Theta(n)$. За всяка конкретна стойност на разделителя броят на взривовете се определя с обхождане на масива от височините на сградите: започваме да събаряме сградите от единия или другия край на редицата според правилото от предишния абзац; продължаваме събарянето, докато стигнем до разделителя; после започваме да събаряме сградите от другия край на редицата. Така за всяка отделна стойност на разделителя е нужно време $\Theta(n)$ за пресмятане на броя на взривовете. Общото време за всички стойности на разделителя е $\Theta(n) . \Theta(n) = \Theta(n^2)$, което надхвърля горната граница $O(n)$ от условието на задачата, т.е. този алгоритъм е бавен.

По-бърз алгоритъм ще получим, като спестим сметките, които се повтарят. Например броят на взривовете при разделител k се различава малко от броя им при разделител $k + 1$. По-конкретно, нека Left[0...n] е масив от цели неотрицателни числа, Left[k] е броят взривове, нужни за събаряне отляво на сградите с номера $1, 2, \ldots, k$. Масива Left[0...n] попълваме в нарастващ ред на индексите по следните формули:

$$\text{Left}[0] = 0; \quad \text{Left}[1] = 1; \quad \text{Left}[k] = \text{Left}[k-1] + \begin{cases} 0, \text{ ако } k \geq 2 \text{ и } A[k-1] > A[k]; \\ 1, \text{ ако } k \geq 2 \text{ и } A[k-1] \leq A[k]. \end{cases}$$

Последното събираемо (0 или 1) изразява факта, че докато височината на сградите намалява при движение отляво надясно, няма нужда от нов взрив (събираемо 0); нов взрив е нужен само когато височината на сградите расте или остава същата (събираемо 1).

Аналогично, Right[0...n] е масив от цели неотрицателни числа, Right[k] е броят взривове, нужни за събаряне отдясно на сградите с номера $k+1$, $k+2$, . . . , n. Масива Right[0...n] попълваме в намаляващ ред на индексите по следните формули:

$$\text{Right}[n] = 0;\ \ \text{Right}[n-1] = 1;\ \ \text{Right}[k] = \text{Right}[k+1] + \begin{cases} 0, & \text{ако } k \le n-2 \text{ и } A[k+2] > A[k+1]; \\ 1, & \text{ако } k \le n-2 \text{ и } A[k+2] \le A[k+1]. \end{cases}$$

Последното събираемо (0 или 1) изразява факта, че докато височината на сградите намалява при движение отдясно наляво, няма нужда от нов взрив (събираемо 0); нов взрив е нужен само когато височината на сградите расте или остава същата (събираемо 1).

Следователно сборът Left[k] + Right[k] е равен на броя на взривовете при разделител k. Най-малкият от тези сборове е точно търсеният минимум.

Тази идея — да бъдат решени "по-малките" задачи преди голямата задача — е известна в теорията на алгоритмите като ***динамично програмиране.*** На практика то означава програмиране на рекурентни формули.

Псевдокод:

```
ALG_1(A[1...n]: array of positive integers)
Left[0...n], Right[0...n]: arrays of non-negative integers
Left[0] ← 0
Left[1] ← 1
for k ← 2 to n do
  if A[k-1] > A[k]
    Left[k] ← Left[k-1]
  else
    Left[k] ← Left[k-1] + 1
Right[n] ← 0
Right[n-1] ← 1
for k ← n-2 downto 0 do
  if A[k+2] > A[k+1]
    Right[k] ← Right[k+1]
  else
    Right[k] ← Right[k+1] + 1
minExplosions ← Left[0] + Right[0]
splitter ← 0
for k ← 1 to n do
  current ← Left[k] + Right[k]
  if current < minExplosions
    minExplosions ← current
    splitter ← k
return minExplosions, splitter
```

Времето на всеки от трите последователни цикъла, а значи и на целия алгоритъм е $\Theta(n)$. Допълнителната памет за масивите Left и Right също е $\Theta(n)$, т.е. изискванията са изпълнени.

Пример: За данните от условието на задачата алгоритъмът се изпълнява така:

k	0	1	2	3	4	5	6	7	8	9	10	$11=n$
$A[1...n]$	—	20	10	8	9	15	15	17	14	15	16	17
Left	0	1	1	1	2	3	4	5	5	6	7	8
Right	5	4	3	3	3	2	2	1	1	1	1	0
сборове	5	5	4	4	5	5	6	6	6	7	8	8

Най-малкият сбор Left[k] + Right[k] е равен на 4 и се достига при $k = 2$ и при $k = 3$. Тоест за събарянето на всички сгради са нужни най-малко четири взрива и това може да стане по два начина. Първият начин е да изберем разделителя между втората и третата сграда. Понеже $10 > 8$, трябва да съборим първо десните сгради. С един взрив в края на редицата събаряме последните четири сгради — тези с височини 17, 16, 15 и 14 метра. Втори взрив в края на новата редица бута последните две сгради — тези с височини 17 и 15 метра. Трети взрив, отново в края на редицата, събаря три сгради — с височини 15, 9 и 8 метра. Достигнахме разделителя, с което изчерпихме десните взривове. Четвъртият взрив е ляв: поставен в началото на редицата, той бута първите две сгради — с височини 20 и 10 метра. Така всички сгради са бутнати с четири взрива.

Другият начин е да изберем за разделител мястото между третата и четвъртата сграда. Понеже $8 < 9$, бутането на сградите трябва да започне от левия край на редицата. Тоест първият взрив е най-отляво и събаря първите три сгради — с височини 20, 10 и 8 метра. С това разделителят е достигнат, така че следващите взривове са десни. Вторият взрив бута последните четири сгради (17, 16, 15, 14); третият взрив бута още две сгради (17 и 15); четвъртият взрив събаря последните две сгради (с височини 15 и 9 метра).

Така построеният алгоритъм е бърз, но използва паметта доста неикономично. Тъй като търсим най-малък сбор от съответните елементи на двата масива, то достатъчно е да пазим само текущите стойности, а не целите масиви. Единствената причина да пазим двата масива е, че техните елементи се изчисляват в противоположни посоки: единият — отляво надясно; другият — отдясно наляво. Ако успеем да уеднаквим посоките, няма да се налага да пазим двата масива (а само текущите стойности).

Можем да постигнем това, като пресмятаме например елементите на масива Right в нарастващ ред на индексите, тоест като намаляваме с единица, когато е нужно, стойността на текущия елемент. Не знаем началната стойност Right[0], затова избираме произволна начална стойност, например Right[0] = 0. Така някои елементи на масива Right получават отрицателни стойности, но това не пречи: абсолютната грешка е еднаква за всички елементи, същото важи за сборовете им с елементите на масива Left, затова най-малкият сбор остава на същото място, променя се само сборът. Тоест алгоритъмът правилно намира разделителя, но не и най-малкия брой взривове. Обаче и този недостатък може да се поправи с помощта на последния елемент на масива Right: ако $\text{Right}[n] = 0$, няма грешка; ако пък $\text{Right}[n] \neq 0$, то достатъчно е да извадим стойността на $\text{Right}[n]$ от сбора и грешката ще бъде поправена.

Пример: За данните от условието на задачата алгоритъмът се изпълнява така:

k	0	1	2	3	4	5	6	7	8	9	10	$11=n$
$A[1...n]$	—	20	10	8	9	15	15	17	14	15	16	17
Left	0	1	1	1	2	3	4	5	5	6	7	8
Right	0	–1	–2	–2	–2	–3	–3	–4	–4	–4	–4	–5
сборове	0	0	–1	–1	0	0	+1	+1	+1	+2	+3	+3

Най-малкият сбор Left[k] + Right[k] е равен на –1 и се достига при $k = 2$ и при $k = 3$. Тоест най-добрите разделители се получават правилно (на същите места като преди), обаче броят на взривовете е грешен (–1 вместо 4). От грешната стойност –1 изваждаме Right[n] = –5: (–1) – (–5) = 4, което е верният резултат.

Псевдокод:

```
ALG_2(A[1...n]: array of positive integers)
Left, Right: integers
minExplosions ← 0
splitter ← 0
Left ← 1
Right ← 0
for k ← 1 to n-1 do
  if A[k+1] ≤ A[k]
    Right ← Right - 1
  current ← Left + Right
  if current < minExplosions
    minExplosions ← current
    splitter ← k
  if A[k+1] ≥ A[k]
    Left ← Left + 1
Right ← Right - 1
current ← Left + Right
if current < minExplosions
  minExplosions ← current
  splitter ← k
return minExplosions - Right, splitter
```

Единственият цикъл определя времето на алгоритъма: то е $\Theta(n)$. Вече няма локални масиви, а само променливи от примитивен тип, затова количеството допълнителна памет е $\Theta(1)$. Тоест този алгоритъм изразходва малко време и малко памет. Подобрение на сложността по порядък е невъзможно: сложността по памет бездруго има най-малкия възможен порядък, а сложността по време в тази задача не може да е сублинейна, защото входните данни са неструктурирани, т.е. алгоритъмът трябва да прочете целия вход (тривиална долна граница по размера на входа).