

**Зад. 1** Даден е масив  $A[1, \dots, n]$  от цели числа, където  $n \geq 1$ . За всяко  $j$  и за всяко  $k$ , такива че  $1 \leq j \leq k \leq n$  казваме, че подмасивът  $A[j, \dots, k]$  е *група* тогава и само тогава, когато са изпълнени следните три условия:

$$\begin{aligned} &\forall i \in \{j, j+1, \dots, k-1\} : A[i] \leq A[i+1] \\ &j = 1 \vee (j > 1 \wedge A[j-1] > A[j]) \\ &k = n \vee (k < n \wedge A[k] > A[k+1]) \end{aligned}$$

*Дължината* на групата  $A[j, \dots, k]$  е  $k - j + 1$ . Групата  $A[j_1, \dots, k_1]$  е *вляво* от групата  $A[j_2, \dots, k_2]$  тогава и само тогава, когато  $k_1 < j_2$ . Очевидно може да има няколко групи с максимална дължина. *Максималната група* е най-лявата група измежду групите с максимална дължина.

**А)** Предложете итеративен алгоритъм със сложност  $O(n)$ , който намира максималната група. Входът да бъде масивът  $A[1, \dots, n]$ , а изходът да бъде наредена двойка индекси  $(j, k)$ , такива че  $A[j, \dots, k]$  е максималната група.

**Б)** Докажете коректността на предложения алгоритъм чрез инварианта.

**Решение:** Задачата може да се реши с алгоритъм, който извършва едно единствено сканиране на масива отляво надясно. Този алгоритъм прилича твърде много на алгоритъма за намиране на мода в сортиран масив, който разгледахме на лекции и който е описан в лекционните записки на курса в муудъл.

Първо да уточним с прости думи какво се иска. Търси се най-дълъг сортиран подмасив. Ако има няколко такива, търси се най-левият от тях. Естествено,  $A[1 \dots n]$  не е непременно сортиран, иначе отговорът би бил тривиален – самият масив, така че не трябва да се извършва *presorting* (за разлика от задачата за намиране на мода).

Всеки елемент сам по себе си е сортиран подмасив, макар и не непременно максимален (както глобално, така и по включване), затова има смисъл сканирането да започне от втория елемент, ако има такъв. За всеки новоизследван елемент  $A[i]$  от масива има точно четири възможности, за които може да мислим като за четири различни състояния (също както в Наблюдение 3 в лекционните записки):

- $A[i]$  може да разширява този максимален сортиран подмасив, който сме открили досега,
- $A[i]$  може да разширява сортиран подмасив, който е по-къс от досегашния най-дълъг и остава такъв с добавянето на  $A[i]$ , но евентуално може да се превърне в най-дълъг,
- с добавянето на  $A[i]$  най-десният максимален по включване сортиран подмасив се превръща в глобално най-дълъг сортиран подмасив в  $A[1 \dots i]$ .
- $A[i]$  започва нов сортиран подмасив, което е същото като  $A[i-1] > A[i]$ .

Следният алгоритъм, който решава задачата, прилича на COMPUTE MODE1 от лекционните записки. Алгоритъмът може да се оптимизира за бързодействие, но не и в асимптотичния смисъл, и тъй като целта е да е лесен за разбиране, е написан по начин, който води до излишни операции.

```

ALG1(A[1,2,...,n])
1  j ← 1, k ← 1
2  jj ← 1, kk ← 1
3  for i ← 2 to n
4      if A[i - 1] ≤ A[i] and (j, k) = (jj, kk)
5          k++, kk++
6      else if A[i - 1] > A[i]
7          jj ← i, kk ← i
8      else
9          kk++
10         if kk > k
11             j ← jj, k ← kk
12  return mode

```

Доказателството за коректност е подобно на това на COMPUTE MODEL от лекционните записки. Естествено, поради ограничението по време на изпита, тук не се очаква толкова подробно доказателство.

**Зад. 2** Решете следните рекурентни уравнения:

$$\begin{array}{ll}
 \text{а) } T(n) = (\log_3 7)T\left(\frac{n}{\log_2 5}\right) + n & \text{б) } S(n) = S(n-2) + \frac{1}{n} \\
 \text{в) } R(n) = 2R(n-1) + \sum_{k=0}^n \binom{n}{k} & \text{г) } Q(n) = Q\left(\frac{n}{2}\right) + Q\left(\frac{n}{3}\right) + n
 \end{array}$$

**Решение:**  $T(n) = \Theta(n)$  от първи случай на МТ. Достатъчно е да съобразите, че  $\log_3 7 < 2$ , но  $\log_2 5 > 2$ .

$S(n) = \Theta(\lg n)$ . Може да се реши като се съобрази, че  $S_1(n) = S_1(n-1) + \frac{1}{n}$  има решение  $S_1(n) = \Theta(\lg n)$  и после в  $S(n) = S(n-2) + \frac{1}{n}$  да се замести  $n$  с  $2m$  и да се извади множител  $\frac{1}{2}$  пред скоби.

$R(n) = \Theta(n2^n)$ . Достатъчно е да съобразите, че сумата от биномните коефициенти е точно  $2^n$  и да приложите метода с характеристичното уравнение.

$Q(n) = \Theta(n)$ . Може да се реши чрез дърво на рекурсията. Пример за подобно решение има в сборника, само че там коефициентите пред  $n$  в двете събираеми е точно единица, а тук е по-малък от единица.

**Зад. 3** Трябва да се напечата красиво текст върху страница. Текстът се състои от  $n$  думи  $w_1, \dots, w_n$ , в този ред, с дължини съответно  $\ell_1, \dots, \ell_n$ . Използва се шрифт тип *monospace*, тоест всички букви, както и шпациите между думите, са с една и съща ширина (думите не съдържат шпации). Всеки ред на страницата съдържа точно  $m$  символа, които може да са буквите от азбуката или шпации. Текстът няма нищо друго освен въпросните думи и шпациите между тях; с други думи, няма пунктуация. Всяка от думите е непразна (очевидно) и може да се побере на един ред; с други думи,  $\forall i: 1 \leq \ell_i \leq m$ . Страницата е достатъчно дълга – можете да допуснете, че има поне  $n$  реда и дори всяка дума да е на отделен ред, няма да има проблеми да бъде побран текстът. Празни редове преди текста и в текста няма. Празните редове след текста нямат значение и не ни интересуват.

Текстът трябва да бъде ляво подравнен (това означава, че всеки ред, на който има думи, започва с най-лявата буква на най-лявата дума) и между всеки две думи на един ред трябва да има точно една шпация. Лесно се вижда, че при това ограничение, ако на даден ред са написани думите  $w_j, \dots, w_k$ , където  $1 \leq j \leq k \leq n$ , то редът ще завършва с точно  $m - \left(\sum_{i=j}^k \ell_i\right) - (k-j)$  шпации, които запълват мястото от най-дясната буква на най-дясната дума  $w_k$  до края на реда. Това “ $(k-j)$ ” е заради шпациите между думите. Съвкупността от всички шпации в края на редовете, които имат текст, се нарича *бялото поле* (*whitespace* на английски).

Да бъде напечатан текстът красиво означава да се минимизира бялото поле в следния смисъл. В идеалния случай бяло поле няма и тогава текстът изглежда най-добре, защото всеки ред завършва с буква, а не с шпация. Но в някои случаи (на практика, най-вероятно) бялото поле е неизбежно – това зависи от дължините на думите и ширината на страницата. Ние искаме не просто бялото поле като цяло да е колкото е възможно по-малко, а особено държим да няма редове с много шпации в края. За

тази цел минимизираме сумата от **квадратите** на броя на шпациите в края на редовете, които имат текст. Същото нещо, написано формално: ако текстът е разположен върху  $t$  реда и ред  $i$ , за  $1 \leq i \leq t$ , завършва с точно  $g(i)$  шпации, то искаме сумата  $\sum_{i=1}^t (g(i))^2$  да е минимална.

Предложете ефикасен алгоритъм, който разполага текста на страницата така, че тази сума да е минимална. Достатъчно е да изчислите само цената на оптималното решение (а не самото разполагане на думите по редове).

**Решение:** Тази задача далечно напомня на задачата за хората на опашка за билети, които могат да се съчетават по групи от двама или да минават поотделно. Тук нещата са по-сложни, тъй като едно групиране (на думи от текста) може да е от повече от две думи.

Наивното решение е да се разгледат всички възможни групирания на думи заедно **без** невъзможните. А невъзможните са тези, при които има групиране на думи, чиято обща дължина плюс броят на шпациите между тях надхвърля дължината на един ред (невъзможни може и да няма). Броят на групиранията в най-лошия случай е от порядъка на  $2^n$ , така че наивното решение (с брутална сила) е абсолютно непрактично.

Разсъждаваме така. Нека  $\text{opt}(k)$  е оптималната (минималната) цена за разполагане на думите  $w_1, \dots, w_k$ , в този ред. Очевидно  $\text{opt}(1) = m - \ell_1$ , а търсеното число е  $\text{opt}(n)$ . За да получим ефикасен алгоритъм, трябва  $\text{opt}(k)$  да се пресмята чрез вече намерени (и запомнени в таблица) стойности  $\text{opt}(j)$  за  $j < k$ . И така, за всяко  $k$  разглеждаме всички  $j$ , такива че  $1 \leq j < k$  и думите  $w_{j+1}, \dots, w_k$ , в този порядък, може да се съберат на един ред (с шпации помежду им, естествено). Това означава, че се разглеждаме възможността  $w_1, \dots, w_j$ , в този порядък, да са разположени на някакви последователни редове (няма значение колко, това не ни интересува) и на следващия ред да са  $w_{j+1}, \dots, w_k$ , в този порядък. Цената за разполагането на  $w_1, \dots, w_j$ , в този порядък, е  $\text{opt}(j)$ . При изчисляването вече я имаме, тъй като  $j < k$ . Тогава цената  $\text{opt}(k)$  е

$$\text{opt}(k) = \min_{\substack{j \in \{1, \dots, k-1\} \text{ и} \\ w_{j+1} \dots w_k \text{ се побират на един ред}}} \left\{ \text{opt}(j) + \left( m - \left( \sum_{i=j+1}^k \ell_i \right) - (k - j - 1) \right)^2 \right\} \quad (1)$$

Тази рекурентна зависимост на пръв поглед води до кубичен алгоритъм, тъй като пресмятането на сумата може да иска линейно време. Сумата е необходима, за да намерим броя на шпациите в края на последния ред. Следният трик ни дава възможност да се отървем от сумата и да пресмятаме броя на шпациите в края на последния ред в константно време. Нека  $S_i = \sum_{t=1}^i \ell_t$ , за  $1 \leq i \leq n$ . Стойностите  $S_i$  може да бъдат изчислени и запомнени предварително. Тогава  $\left( \sum_{i=j+1}^k \ell_i \right) - (k - j - 1) = S_k - S_j$ . Нещо повече, условието  $w_{j+1}, \dots, w_k$  да се побират на един ред е същото като  $S_k - S_j \leq m$ . Тогава от (1) имаме

$$\begin{aligned} \text{opt}(1) &= m - \ell_1 \\ \text{opt}(k) &= \min_{j \in \{1, \dots, k-1\} \wedge S_k - S_j \leq m} \left\{ \text{opt}(j) + (m - (S_k - S_j))^2 \right\} \end{aligned} \quad (2)$$

Оттук директно имаме  $O(n^2)$  алгоритъм, който решава задачата.

**Зад. 4** Даден е ориентиран граф  $G = (V, E)$ . Дефинирана е релацията  $R \subseteq V \times V$  така:

$\forall u, v \in V : (u, v) \in R$  тогава и само тогава, когато съществува ориентиран път (маршрут) от  $u$  до  $v$  и от  $v$  до  $u$

Очевидно,  $R$  е релация на еквивалентност. Предложете ефикасен алгоритъм, който получава на входа описание на графа чрез списъци на съседство и връща класовете на еквивалентност на  $R$ . Съвсем накратко обосновайте коректността на предложението от Вас алгоритъм и кажете каква е сложността му по време.

**Решение:** Става дума за силно свързаните компоненти на графа. И то не за самите компоненти, а само за върховете им. Алгоритъмът е изучаван на лекции.