Примерни решения на домашно № 3

Задача 1: Нека $A[1..n]$ и $B[1..n]$ са сортирани масиви. За удобство може да допуснете, че никой от тях няма повтарящи се елементи и те нямат общи елементи. Докажете, че $2n-1$ е долна граница за броя на сравненията при сливането (*merge*) на тези два масива в един сортиран масив $C[1..2n]$.

Решение: Да допуснем, че съществува алгоритъм AlgX, такъв че AlgX$(A[1..n], B[1..n])$ връща $C[1..2n]$, който се състои от елементите на A и B, но в сортиран вид, като AlgX ползва най-много $2n-2$ сравнения за целта. Тъй като C има точно $2n$ елемента, съседните двойки елементи в C са точно $2n-1$. Щом са ползвани най-много $2n-2$ сравнения, съществува двойка съседни в C елементи, които не са били сравнени от алгоритъма.

Нека A и B са такива, че

$$A[1] < B[1] < A[2] < B[2] < \cdots < A[n] < B[n]$$

Тогава в C се редуват елемент от A с елемент от B:

$$C = [A[1], B[1], A[2], B[2], \ldots, A[n], B[n]]$$

Тук няма загуба на общност, понеже, ако долната граница е в сила за такива A и B, тя е в сила и изобщо.

Щом $C = [A[1], B[1], A[2], B[2], \ldots, A[n], B[n]]$ и двойка съседи в C не са били сравнени, поне едното от следните е вярно.

- $A[i]$ не е бил сравнен с $B[i]$, за някое $i \in \{1, \ldots, n\}$. Тогава противникът прави $A[i]$ по-голям от $B[i]$, но така, че $A[i] < A[i+1]$ да остане вярно. Спрямо тази промяна на входа, изходът би трябвало да бъде

  $$[A[1], B[1], A[2], B[2], \ldots, A[i-1], B[i-1], B[i], A[i], A[i+1], B[i+1], \ldots, A[n], B[n]]$$

  Но AlgX продължава да връща $[A[1], B[1], A[2], B[2], \ldots, A[n], B[n]]$, понеже резултатите от останалите сравнения са същите като тези преди промяната на входа. Този масив вече не е коректен изход. По този начин противникът опровергава AlgX, без да може да бъде уличен в лъжа.

- $A[i]$ не е бил сравнен с $B[i-1]$, за някое $i \in \{2, \ldots, n\}$. Тогава противникът прави $B[i-1]$ по-голям от $A[i]$, но така, че $B[i-1] < B[i]$ да остане вярно. Спрямо тази промяна на входа, изходът би трябвало да бъде

  $$[A[1], B[1], A[2], B[2], \ldots, A[i-1], A[i], B[i-1], B[i], A[i+1], B[i+1], \ldots, A[n], B[n]]$$

  Но AlgX продължава да връща $[A[1], B[1], A[2], B[2], \ldots, A[n], B[n]]$, понеже резултатите от останалите сравнения са същите като тези преди промяната на входа. Този масив вече не е коректен изход. По този начин противникът опровергава AlgX, без да може да бъде уличен в лъжа.

Заключаваме, че такъв AlgX не съществува.

Задача 2: Дадена е непразна редица от цели положителни числа $\alpha = \langle a_1, a_2, \ldots, a_n \rangle$. Дадено е и цяло число $k \in \{1, \ldots, n\}$. Нека i_0, i_1, i_2, $\ldots$, i_{k-1}, i_k са числа, такива че

$$\begin{aligned} i_0 &= 1, \\ i_0 &< i_1, \\ i_1 &< i_2, \\ &\ldots \\ i_{k-1} &< i_k, \\ i_k &= n+1 \end{aligned}$$

От тях i_0 и i_k са фиксирани, а останалите наричаме *индексите*. Нека I е множеството от индексите. Цената на I е

$$c(I) = \max_{0 \le j \le k-1} \sum_{s=i_j}^{i_{j+1}-1} a_s$$

Предложете ефикасен алгоритъм, който намира такива индекси, че $c(I)$ е минимална. Достатъчно е Вашият алгоритъм да намира само цената, а не самите индекси. Аргументирайте коректността на Вашия алгоритъм и изследвайте сложността му по време.

Упътване: Помислете за алгоритъм по схемата ***Динамично Програмиране.***

Решение: На прост български, дадена е редица от n положителни числа, примерно

$$\alpha = \langle 100, 50, 40, 25, 200, 55, 95 \rangle$$

и цяло положително $k \le n$. Иска се α да бъде разбита на k непрекъснати подредици, такива че максималната сума от елементи на подредица да е минимална. Можете да мислите, че елементите на α са някакви работи (jobs), като всяка работа е дефинирана чрез броя часове за нейното извършване, и се иска да се разпределят тези работи между k човека най-равномерно – по такъв начин, че максималното натоварване на човек да се минимизира. Съществено ограничение е, че работите в α са подредени линейно поначало, тази наредба е фиксирана и всеки работник получава работи, които са непрекъсната подредица в α. Без последното ограничение, задачата е **NP**-трудна дори за $k = 2$. С последното ограничение обаче тя е решима ефикасно чрез алгоритъм по схемата **Динамично Програмиране**.

Забележете, че това има смисъл дори за $k = 1$. Съгласно формалното описание, тогава $I = \emptyset$, като цената не е нула, а е

$$c(\emptyset) = \sum_{s=i_0}^{i_{0+1}-1} a_s = \sum_{s=1}^{i_1-1} a_s = \sum_{s=1}^{n} a_s$$

тоест, сумата от елементите на α. Което е смислено, ако всички работи се дават на един работник.

Ако $k = 2$, задачата е елементарна. За примера с $\alpha = \langle 100, 50, 40, 25, 200, 55, 95 \rangle$ на око се вижда, че оптималното разделение на работите е $100, 50, 40, 25$ на единия и

$200, 55, 95$ на другия, като първата редица има сума 215, втората има сума 350, така че цената на това решение е 350.

За произволно $n \geq 2$ и $k = 2$, решението е минимумът от следните $n - 1$ числа:

$$
\begin{aligned}
&\max\{a_1,\, a_2 + a_3 + a_4 + \cdots + a_{n-1} + a_n\}\\
&\max\{a_1 + a_2,\, a_3 + a_4 + \cdots + a_{n-1} + a_n\}\\
&\max\{a_1 + a_2 + a_3,\, a_4 + \cdots + a_{n-1} + a_n\}\\
&\ldots\\
&\max\{a_1 + a_2 + a_3 + \cdots + a_{n-1},\, a_n\}
\end{aligned}
$$

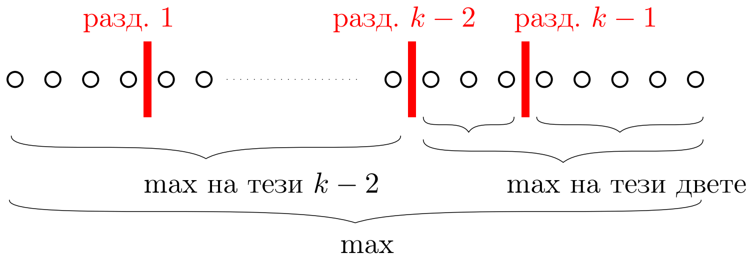
Всяко от тези числа се получава след нещо като слагане на един разделител в някоя "празнина" между две числа в редицата, като празнините са $n-1$ на брой, и вземане на максимума от двете суми спрямо разделителя.

За големи стойности на n и k обаче решенията с груба сила стават неефикасни. Всяко решение се определя еднозначно от разполагането на $k-1$ разделители, отговарящи на индексите, между елементите на редицата, като обаче не може да има два разделителя един до друг. Накратко, има $n-1$ позиции и от тях се избират $k-1$, на които се разполагат разделители. Това прави $\binom{n-1}{k-1}$ начина. Както знаем, при горен индекс n, средният биномен коефициент има асимптотика, близка до $\Theta(2^n)$, така че решение с груба сила е безполезно на практика при $k \approx n/2$.

Разсъждаваме така. Нека $k \geq 2$. Последната редица започва веднага след най-десния разделител. Тогава характеристика на всяко решение, оптимално или не, е мястото на най-десния разделител. Да си представим **фиксирани разделители**, които са сложени последователно отляво надясно. Тогава най-десният разделител е сложен последен. Преди неговото слагане е имало $k - 1$ редици, разделени чрез $k - 2$ разделителя. Цената на решението, съдържащо най-десния разделител, е максимумът на следните:

- максималната сума на редица измежду първите $k - 2$ редици и
- максималната сума на някоя от последните две редици, получени от бившата последна редица чрез слагане на най-десния разделител някъде в нея.

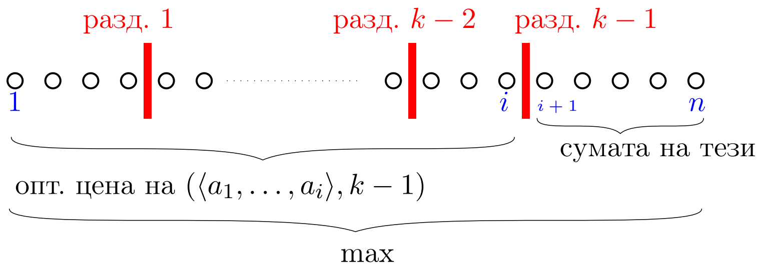
Образно:



Да помислим за рекурсивна декомпозиция. Екземпляр на задачата е наредена двойка от числена редица и брой на разделителите. Подредицата вляво от най-десния разделител и числото $k-1$ (броят на подредиците в нея) е екземпляр на задачата с по-къса числена редица и по-малко число от оригинала, така че може да направим рекурсивна декомпозиция. За фиксирана позиция на най-десния разделител—да кажем, че той е между i-ия и $(i+1)$-вия елемент—оптималната цена на решение е максимумът от

- оптималната цена на екземпляра вляво и
- сумата $a_{i+1} + \cdots + a_n$.

Принципът на оптималността е в сила: няма смисъл да разглеждаме не-оптимално решение на екземпляра вляво. Образно:



Това i може да е най-малко $k-1$ и най-много $n-1$. Тогава оптималната цена на оригиналния екземпляр е

$$M(n,k) = \min \left\{ \max \left\{ M(i,k-1), \sum_{j=i+1}^{n} a_j \right\} : i \in \{k-1, \ldots, n-1\} \right\}$$

Да видим началните условия. Те са

$$M(m,1) = \sum_{s=1}^{m} a_s$$
$$M(m,m) = \max \{a_1, \ldots, a_m\}$$

за $m \in \{1, \ldots, n\}$.

Изчислението става с "трапецовидна" таблица. Нека масивът е $M[1\mathinner{..}n, 1\mathinner{..}k]$. Ползва се само частта под и включително главния диагонал, откъде и името "трапецовидна".

В колона 1, отгоре надолу се слагат a_1, $a_1 + a_2$, …, $a_1 + \cdots + a_n$. В главния диагонал, в посока от горе-ляво към долу-дясно, се слагат a_1, $\max\{a_1, a_2\}$, …, $\max\{a_1, a_2, \ldots, a_n\}$. Това са началните условия.

Отговорът, който ни трябва, е $M[n,k]$. Той се пресмята като минимум на $n-1-(k-1)+1 = n-k+1$ числа, всяко от които е максимум от две числа, едното от които вече е записано в клетка от $M[n-1,k-1]$, $M[n-2,k-1]$, …, $M[k-1,k-1]$ (общо $n-k+1$ числа), а другото е една сума.

В общия случай, за $3 \leq p \leq n$ и $2 \leq q \leq p-1$:

$$M[p,q] \leftarrow \min_{q-1 \leq i \leq p-1} \max \{M[i,q-1], a_{i+1} + \cdots + a_n\}$$

Редът на изчислението може да е подобен на този, който приехме за триъгълника на Паскал: по редове отгоре надолу, в рамките на един ред до главния диагонал – отляво надясно.

Ето илюстрация на изчислението на $M[n, k]$ в конкретен пример. Червената клетка $M[10, 6]$ се изчислява чрез сините клетки $M[9, 5]$, $M[8, 5]$, $M[7, 5]$, $M[6, 5]$ и $M[5, 5]$.

$n \backslash k$	1	2	3	4	5	6
1	a_1					
2	$a_1 + a_2$	max a_1, a_2				
3	sum $a_1 \mathinner{..} a_3$		max $a_1 \mathinner{..} a_3$			
4	sum $a_1 \mathinner{..} a_4$			max $a_1 \mathinner{..} a_4$		
5	sum $a_1 \mathinner{..} a_5$				max $a_1 \mathinner{..} a_5$	
6	sum $a_1 \mathinner{..} a_6$					max $a_1 \mathinner{..} a_6$
7	sum $a_1 \mathinner{..} a_7$					
8	sum $a_1 \mathinner{..} a_8$					
9	sum $a_1 \mathinner{..} a_9$					
10	sum $a_1 \mathinner{..} a_{10}$					

На свой ред те ползват $M[8, 4]$, $M[7, 4]$, $M[6, 4]$, $M[5, 4]$ и $M[4, 4]$, и така нататък.

$n \backslash k$	1	2	3	4	5	6
1	a_1					
2	$a_1 + a_2$	max a_1, a_2				
3	sum $a_1 \mathinner{..} a_3$		max $a_1 \mathinner{..} a_3$			
4	sum $a_1 \mathinner{..} a_4$			max $a_1 \mathinner{..} a_4$		
5	sum $a_1 \mathinner{..} a_5$				max $a_1 \mathinner{..} a_5$	
6	sum $a_1 \mathinner{..} a_6$					max $a_1 \mathinner{..} a_6$
7	sum $a_1 \mathinner{..} a_7$					
8	sum $a_1 \mathinner{..} a_8$					
9	sum $a_1 \mathinner{..} a_9$					
10	sum $a_1 \mathinner{..} a_{10}$					

Клетките под петия диагонал реално не се ползват. Дали ще ги пресметнем или не, зависи от това, доколко искаме да оптимизираме изчислението. Асимптотиката в най-лошия случай не се променя (дори да запълним цялата правоъгълна таблица и над главния диагонал, асимптотиката на най-лошия случай не се променя).

Ето алгоритимът. Записът не е съвсем детайлен, но е ясно как да се имплементира.

Разбиване на подредици($\alpha = \langle a_1, \ldots, a_n \rangle, k \in \{1, \ldots, n\}$)

създай $M[1 \mathinner{..} n, 1 \mathinner{..} k]$
for $i \leftarrow 1$ **to** n
    $M[i,1] \leftarrow a_1 + \cdots + a_i$
for $i \leftarrow 2$ **to** n
    $M[i,i] \leftarrow \max(a_1, \ldots, a_i)$
for $p \leftarrow 3$ **to** n
    for $q \leftarrow 2$ **to** $p-1$
        $M[p,q] \leftarrow \min_{q-1 \leq i \leq p-1}(\max(M[i, q-1], a_{i+1} + \cdots + a_n))$
return $M[n,k]$

Коректността следва директно от коректността на рекурсивната декомпозиция. Сложността по време, ако се имплементира буквално, е $\Theta(kn \times n^2)$, което е $\Theta(kn^3)$. Причината е, че клетките са $\Theta(kn)$ и времето за попълване на клетка $M[p,q]$ е $\Theta((p-q) \times (n-p+q))$, което е $\Theta(n^2)$ в най-лошия случай.

Сложността може да бъде подобрена до $\Theta(kn^2)$, ако сумите $a_{i+1} + \cdots + a_n$ не се изчисляват буквално по този начин, а чрез предварително пресмятане на стойностите на суфиксите и съхраняването им в едномерен масив (друг алгоритъм по схемата **Динамично Програмиране**.)