Михаил Захариев Захов, фн: 45655

Зад. 1. Решение:

Ще докажа, че за алгоритъм, който слива два сортирани масива с дължина n в сортиран масив с дължина 2n, където елементите масивите са без повтарящи се и без общи елементи, и който прави по-малко от 2n-1 на брой сравнения за всеки свой вход, съществуват нерадостни входове. Щом има нерадостни входове, то този алгоритъм няма как да е коректен. От това следва, че всеки коректен алгоритъм поне за един вход трябва да направи поне 2n-1 на брой сравнения, т.е 2n-1 ще е долна граница за броя на сравненията при сливането.

Нека $n = 1$. Тогава очевидно алгоритъмът ще трябва да направи поне $1 = 2.1 - 1$ на бр. сравнения, за да определи или $a_1 < b_1$, или $b_1 < a_1$.

Нека $n \geq 2$.

Какви сравнения има смисъл да прави алгоритъмът?

1. Безсмислено е да прави сравнения от вида $a_i < a_j$ и $b_i < b_j$, защото по условие знаем, че A и B са сортирани. Тези сравнения не ни дават нова информация, ненужно увеличават броя на сравненията и всеки ефективен алгоритъм може да се модифицира да работи без такива сравнения.

2. Сравнението $a_i < b_j$ ни дава общата информация като сравненията: $a_i > b_j$, $b_j < a_i$, $b_j > a_i$. Имаме:

$a_i < b_j$ е T/F $\Leftrightarrow$ $a_i > b_j$ е F/T $\Leftrightarrow$

$b_j < a_i$ е F/T $\Leftrightarrow$ $b_j > a_i$ е T/F

(Заб. По условие $a_i \neq b_j$, $\forall i,j \in \{1 \dots n\}$).

Но щом тези ~~двойки~~ сравнения са еквивалентни, то всеки ефективен алгоритъм може да се модифицира така, че да ползва само сравнения от вида $a_i < b_j$.

Пр.: Ако имаме if($a_i > b_j$){…}, можем да го променим на if(!$a_i < b_j$){…} и двете ще са екв.

Следващите разсъждения са за алгоритъм в "нормална форма" със сравнения единствено във вида $a_i < b_j$.

Допускаме, че алгоритъмът Alg е коректен и прави $m \leq 2n-2$ на брой сравнения за всеки вход.

~~Всяка~~ Сравненията, които прави Alg за някой вход изглеждат така:

(1)
$$\begin{cases} a_{i_1} < b_{j_1} \\ a_{i_2} < b_{j_2} \\ \vdots \\ a_{i_m} < b_{j_m} \end{cases}, \quad i_k, j_k \in \{1 \dots n\}, \quad m \leq 2n-2$$

1сл. $\exists q \in \{1 \dots n\}$, т.ж b_q не участва в нито едно сравнение. Тогава входовете:

$\dots < b_{q-1} < a_q < b'_q < \dots$ и

$\dots < b_{q-1} < b''_q < a_q$, които се различават само по числото в масива B на q-та позиция, са неразличими за алгоритъма. Наистина, тъй като $A' = A''$ и $B' = B''$ с изключение на q-тата позиция в B', B'', четността на сравнението (1) се запазва (не сме променили нито едно от числата, които участват в сравнението). Алгоритъмът трябва да съобрази на едни и същи редица от сравнения (с еднаква четност) два различни изхода. Алгоритъмът не може да "гадае" кой е верният изход, следователно допускането, че е коректен е невярно.

Заб. Тук дали не се налага да ползваме броя на сравненията.

Заб. Интуицията, че ако един алгоритъм не ги чете целия вход, то той не е коректен, се оказва вярна в този случай.

2сл. $\forall q \in \{1 \dots n\}$, b_q участва в поне едно сравнение.

От това следва, че $\exists k, s \in \{1 \dots n\}$, $k \neq s$, т.ж b_k и b_s участват в точно едно сравнение $a_{k_1} < b_k$, $a_{s_1} < b_s$.

Наистина, ако допуснем противното, то тогава поне $n-1$ на брой елементи от B трябва да са в поне две сравнения:

$m \leq 2n-2 < \underbrace{2(n-1)}_{\text{поне } n-1 \text{ тр. да участват в поне 2.}} + \underbrace{1}_{\text{един тр. да участват в поне 1.}}$

Ще докажем чрез, че $k \neq 1$ или $s \neq 1$.

Нека $k \neq 1$.

Разглеждаме входа (A', B'), където:

$A'[1 \dots n] = a_1 a_2 \dots a_n$,

$B'[1 \dots n] = b_1 b_2 \dots b'_k \dots b_n$ и общо,

$b_1 < a_1 < b_2 < a_2 < \dots < a_{k-1} < b'_k < a_k < b_{k+1} < \dots < b_n < a_n$.

(т.е $a_i < b_j$ е вярно $\Leftrightarrow i < j$)

1сл. За входа (A', B') единственото сравнение $a_{k_1} < b'_k$, в което участва b'_k, е истина.

Разглеждаме входа (A'', B''), който се различава от (A', B') единствено в b'_k по този начин: $a_k < b''_k < b_{k+1}$.

Един над друг изглеждат така:

$b_1 < a_1 < \dots < a_{k_1} < \dots < a_{k-1} < b'_k < a_k < b_{k+1} < \dots < b_n < a_n$

$b_1 < a_1 < \dots < a_{k_1} < \dots < a_{k-1} < a_k < b''_k < b_{k+1} < \dots < b_n < a_n$

Редицте от сравнения (1) за входовете (A', B') и (A'', B'') са едни и същи и сравненията са с една и съща истинност.

Всяко сравнение, в което не участват b'_k и b''_k има една и съща истинност, защото елементите им са едни и същи. Сравненията $a_{k_1} < b'_k$ и $a_{k_1} < b''_k$ също са едновременно верни, заради дефиницията на (A', B') и (A'', B''). алгоритъмът

На една и съща редица от сравнения се отговаря да съответства два изхода. Значи алгоритъмът трябва да "познава", кой е правилният изход, следователно е некоректен

2сл. За всяка (A', B') единственото сравнение $a_{k_1} < b'_k$, в което участва b'_k е лъжа.

Този случай е напълно аналогичен на 1сл., като за неразличните входове ще имаме следното: $\dots < a_{k_1} < \dots$

$$b_1 < a_1 < \dots < b_{k-1} < a_{k-1} < b'_k < a_k < \dots < b_n < a_n$$

$$b_1 < a_1 < \dots < b_{k-1} < b''_k < a_{k-1} < a_k < \dots < a_{k_1} < \dots < b_n < a_n$$

Отново редицата от сравненията ще е една и съща с еднаква истинност за тези, в които b'_k и b''_k не участват, и общо $a_{k_1} < b'_k$ и $a_{k_1} < b''_k$ ще са едновременно неверни.

Алгоритъмът трябва да "гадае" $\Rightarrow$ е некоректен.

Заключение

Не може да има коректен алгоритъм за дадената задача, който за всеки вход извършва $\leq 2n-2$ сравнения. Следователно $2n-1$ е долна граница за броя на сравненията при алгоритъм за сливане. $\square$

Зад. 2

Дадена е редица от цели положителни числа $d = \langle a_1 \dots a_n \rangle$, $n \in \mathbb{N}^+$. Дадено е $k \in \{1 \dots n\}$. Да се намери разбиване на d на подредици $\langle a_1 \dots a_{i_1-1} \rangle \langle a_{i_1}, \dots, a_{i_2-1} \rangle \dots \langle a_{i_{k-1}} \dots a_{i_k-1} \rangle$, т.ще $1 = i_0 < i_1 < \dots < i_{k-1} < i_k = n+1$ и максималната сума на елементите на подредица да е минимална.

Реш:

С I_m^r ще означаваме н-бо от индекси $i_1 \dots i_{r-1}$, където $1 \le m \le n$ и $1 \le r \le m$, $r \le k$. $(1 = i_0 < i_1 < \dots < i_{r-1} < i_r = m+1)$.

Т.е с I_m^r ще означаваме н-бото от индексите за подзадачата с редица $d_m = \langle a_1 \dots a_m \rangle$ и дадено число r.

Нека $C_m^r := \min_{\forall I_m^r} \left(c\left(I_m^r\right) \right)$

Т.е с C_m^r ще означаваме минималната сума за подзадачата $d_m = \langle a_1 \dots a_m \rangle$, r. (очевидно C_m^r е число).

I_{1m}^r ще наричаме оптимално н-бо от индекси (за подзадачата $d_m = \langle a_1 \dots a_m \rangle$, r), ако $C_m^r = \min_{\forall I_m^r} \left(c\left(I_m^r\right) \right) = c\left(I_{1m}^r\right)$

(т.е, ако с индексите от I_{1m}^r се постига минималната сума).

Индексите от I_{1m}^r ще наричаме оптимални индекси.

При въведените означения решението на Зад. 2 е:

$$C_n^k = \min_{\forall I_n^k} \left(c(I_n^k) \right).$$

Тв. 1 $C_m^r \le C_{m+1}^r$.

Д-во: Нека $I_{1\,m+1}^r = \{i_1 \dots i_{r-1}\}$ е оптимално m-бо от индекси.

т.е $C_{m+1}^r = \min_{\forall I_m^r} \left(c(I_{m+1}^r) \right) = c\left(I_{1\,m+1}^r \right)$.

$I_{1\,m+1}^r$ разбива редицата $\langle a_1 \dots a_{m+1} \rangle$ така:

$$(1)\ \langle a_1 \dots a_{i_1-1} \rangle \langle a_{i_1} \dots a_{i_2-1} \rangle \dots \langle a_{i_{r-1}} \dots a_{m+1} \rangle$$

1сл. $i_{r-1} < m+1$. Тогава $I_{1\,m+1}^r$ е валидно m-бо от индекси за $\langle a_1 \dots a_m \rangle$ (защото $1 < i_1 < \dots < i_{r-1} < i_r = m+1$).

$I_{1\,m+1}^r$ разбива $\langle a_1 \dots a_m \rangle$ така:

$$(2)\ \langle a_1 \dots a_{i_1-1} \rangle \langle a_{i_1} \dots a_{i_2-1} \rangle \dots \langle a_{i_{r-1}} \dots a_m \rangle.$$

Сумите на първите $r-1$ подредици от (1) и (2) съвпадат, и $\sum_{s=i_{r-1}}^{m} a_s \le \sum_{s=i_{r-1}}^{m+1} a_s$ (по условие $a_j > 0$, $j = 1 \dots n$).

Тогава със сигурност $C_m^r \le C_{m+1}^r$.

2сл. $i_{r-1} = m+1$

Тогава $I_{1\,m+1}^r$ не е валидно m-бо от индекси за редицата $\langle a_1 \dots a_m \rangle$ (защото $i_{r-1} \not< i_r = m+1$), но $I_{1\,m+1}^r \setminus \{i_{r-1}\}$ е.

Получаваме разбиването:

(3) $\langle a_1 \dots a_{i_1-1} \rangle \langle a_{i_1} \dots a_{i_2-1} \rangle \dots \langle a_{i_{r-2}} \dots a_m \rangle$

Очевидно две по две първите $r-1$ подредици от (1) и (3) съвпадат, а защото и сумите на елементите им съвпадат. (защото и max от сумите съвпада).

Бдо в (3) редицата $\langle a_{i_{r-2}} \dots a_m \rangle$ има поне два елемента и защото можем да я разделим така:

$\langle a_{i_{r-2}} \dots a_{i'_{r-1}-1} \rangle \langle a_{i'_{r-1}} \dots a_m \rangle$, то това също поне да намали максимума с на сумите, следователно $C^r_m \le C^r_{m+1}$ □.

Сл. 1

Ако $I^r_{1m} = \{i_1 \dots i_{r-1}\}$, $m < n$ е оптимално, т.е $C^r_m = c(I^r_{1m})$, и $\sum_{s=i_{r-1}}^{m+1} a_s \le C^r_m$, то $C^r_{m+1} = C^r_m = c(I^r_{1m})$.

Малко по-разбираемо: ако добавим елемента a_{m+1} на края на подредицата $\langle a_{i_{r-1}} \dots a_m \rangle$ и $\sum_{s=i_{r-1}}^{m+1} a_s \le C^r_m$, то защото максимума на сумите няма да се промени $C^r_{m+1} = C^r_m$ и разбиването I^r_{1m} ще остане оптимално и за C^r_{m+1}.

Д-во: От тв. 1, тъй като не може $C^r_{m+1} < C^r_m$, защото най-добрата възможна е $C^r_{m+1} = C^r_m$.

По условие $\sum_{s=i_{r-1}}^{m+1} a_s \le C^r_m$, защото прибавянето на a_{m+1} не променя максимума, следователно $C^r_{m+1} = C^r_m$.

Равенството $C^r_{m+1} = C^r_m$ е постигнато при I^r_{1m}, следователно I^r_{1m} остава оптимално за редицата $\langle a_1 \dots a_{m+1} \rangle$. □

Намиране на рекурентната връзка

Обща идея:

Ако знаем C_p^q за някои r, m и, ако знаем $\langle a_{i_{r-1}} \dots a_m \rangle$ за някое оптимално решение за C_m^r, то ще можем да намерим C_{m+1}^r и $\langle a_{i_{r-1}} \dots a_{m+1} \rangle$ (оптимално i_{r-1} за C_{m+1}^r) като разглеждаме единствените възможности за $\langle a_{i_{r-1}} \dots a_{m+1} \rangle$, които са:

$\langle a_{i'_{r-1}} \dots a_{m+1} \rangle$

$\langle a_{i'_{r-1}-1} \dots a_{m+1} \rangle$

$\langle a_{i'_{r-1}-2} \dots a_{m+1} \rangle$

$\vdots$

$\langle a_{m+1} \rangle$.

Формално:

1. Базови случаи (които директно могат да изчислим).

1сл. $C_r^r = \max\limits_{j=1 \dots r} a_j$, $r = 1 \dots k$, защото има едно възможно разбиване и то е: $\langle a_1 \rangle \langle a_2 \rangle \dots \langle a_r \rangle$.

Тъй като разбиването е единствено, то знаем, че $i_{r-1} = r$.

(i_{r-1} е последният оптимален индекс от някое оптимално м-во от индекси).

2сл. $C_m^1 = \sum_{s=1}^{m} a_s$, $m = 1 \dots n$, защото отново имаме единствено разбиване и то е: $\langle a_1 \dots a_m \rangle$.

В този случай $r = 1 \Rightarrow i_{r-1} = i_0 = 1$

Заб. Важно е, че знаем последните оптимални индекси i_{r-1} за базовите случаи, защото ще ни трябват в рекурентната връзка.

2. ЦП.

Нека за някои r и m $(r \le k, m < n)$ знаем C_p^q и съответния му най-голям оптимален индекс i_{q-1}, където $q = 1 \dots r$, $p = 1 \dots m$.

~~Нека, разбира се, r и m са такива, че C_m^r не е~~

3. Стъпка.

Намираме C_{m+1}^r и съответния му най-голям оптимален индекс i_{r-1}.

Нека i'_{r-1} е най-голям оптимален индекс за C_m^r (i'_{r-1} е от някое оптимално разбиване за C_m^r; знаем го от ЦП).

1сл. $\sum_{s=i'_{r-1}}^{m+1} a_s \le C_m^r$. Тогава от Следствие 1 имаме, че оптималното н-во за C_m^r и C_{m+1}^r е едно и също (в частност $i_{r-1} = i'_{r-1}$) и също $C_{m+1}^r = C_m^r$. ✓

ДМ, ср

2сл. $\sum_{s=i'_{r-1}}^{m+1} a_s > C^r_m$. Нека $\sum_{s=i'_{r-1}}^{m+1} a_s = H$.

Тогава $C^r_{m+1} \le H$ (възможно е оптималното н-во за C^r_m да не е оптимално за C^r_{m+1}).

(1) Ако $C^r_{m+1} = H$, то очевидно оптималното н-во I^r_{1m} за C^r_m ще е оптимално и за C^r_{m+1}, защото сумите $\sum_{s=i'_t}^{i'_{t+1}-1} a_s \le C^r_m$, $t = 0 \dots r-2$ и всъщност максималната сума на подредица ще е точно H (сумата на елементите от последната подредица).

Щом I^r_{1m} е оптимално за C^r_{m+1}, то $i_{r-1} = i'_{r-1}$. V

(2) Ако $C^r_{m+1} < H$, то със сигурност $i_{r-1} > i'_{r-1}$ (в нашия случай сумата последната подредица не стига).

Наистина, ако допуснем $i_{r-1} \le i'_{r-1}$, то:

$H = \sum_{s=i'_{r-1}}^{m+1} a_s \le \sum_{s=i_{r-1}}^{m+1} a_s$ (по ус. $a_s > 0$), ще се отделят още

повече от C^r_{m+1}.

Готови сме да напишем рекурентната връзка. Проверяваме няколкото:

$\langle a_{i'_{r-1}}, a_{i'_{r-1}+1}, a_{i'_{r-1}+2}, \dots, a_{m+1} \rangle$,

$\langle a_{i'_{r-1}+1}, a_{i'_{r-1}+2}, \dots, a_{m+1} \rangle$ и $C^{r-1}_{i'_{r-1}}$

$\vdots$

$\langle a_{m+1} \rangle$ и C^{r-1}_m

Получаване уравнението:

$$C^{r}_{m+1} = \min\left\{ \sum_{s=i'_{r-1}}^{m+1} a_s = H,\ \max_{i'_{r-1} \le p \le m} \left\{ \sum_{s=p+1}^{m+1} a_s,\ C_p^{r-1} \right\} \right\}$$

Ако минимумът се достигне в H, то $i_{r-1} = i'_{r-1}$.

Ако се достигне за някое p, то $i_{r-1} = p+1$.

Алгоритъм

В двумерен масив $C[1\dots n][1\dots k]$ ще пазим стойностите C^{r}_{m}, $m = 1\dots n$, $r = 1\dots k$.

(1) В началото $C[1\dots n][1\dots k]$ е празен:

	1	2	3	…	k
1		×	×	…	×
2			×	…	×
3				…	×
⋮	⋮	⋮	⋮	⋱	⋮
n				…	

На местата на ×-овете (т.е. всяка клетка над главния диагонал) няма да записваме нищо (т.е. с m елемента не може да се разбие на $r > m$ непразни м-ва).

(2) После записване базовите случаи, които се намират в клетките на първата колона и по главния диагонал.

(3) Накрая изчисляване колона по колона от горе на долу. Т.е. първо намираме $C^{2}_{3}, C^{2}_{4}, C^{2}_{5} \dots, C^{2}_{n}$, след това $C^{3}_{4}, C^{3}_{5}, \dots, C^{3}_{n}$, и така до $C^{k}_{k+1}, \dots, C^{k}_{n}$.

Find_min ($A[a_1 \dots a_n]$ - непразна редица (масив) от цели положителни числа ; $k \in \{1 \dots n\}$)

```
1.  C[1...n][1...k]   // Двумерен масив, в който ще
2.                    // записваме C_m^r.
3.  C[1][1] ← a1
4.  for j ← 2 to n          // Смятаме C_m^1, m = 1...n.
5.  |  C[j][1] ← C[j-1][1] + aj
6.
7.  max ← -∞
8.  for j ← 1 to k          // Смятаме C_r^r, r = 1...k
9.  |  if aj > max
10. |  |  max ← aj
11. |  C[j][j] ← max
12.
13. // Основната част:
14. for r ← 2 to k
15. |  iop ← r      // Последният оптимален индекс за C_m^r.
16. |  l ← a_r      // Сумата на елементите на последната
17. |               // оптимална подредица за C_m^r.
18. |  for m ← r to n-1
19. |  |  if l + a_{m+1} ≤ C_m^r   // Вж. Сл. 1.
20. |  |  |  C[m+1][r] ← C[m][r]
21. |  |  |  continue
22. |  |
23. |  |  best ← l + a_{m+1}   // досегашен минимум.
24. |  |  sum ← l + a_{m+1}    // Сумата на ел. на последна-
25. |  |                       // та редица.
26. |  |  for t ← iop to m
27. |  |  |  sum ← sum - a_t   // Пробване с по-
28. |  |  |                    // къса последна редица.
29. |  |  |
```

```
30.                 if best ≥ max{sum, C[t][r-1]}
31.                 best ← max{sum, C[t][r-1]}
32.                 iop ← t  //Запазя новия оп. индекс.
33.                 l ← sum  //Зап. новата оп. сума.
34.             C[m+1][r] ← best  //запазя C[m+1][r].
35.
36.  return  C[n][k]
37.
```

Сложност по време

Очевидно редове 1 - 11 се изпълняват за $O(n)$. Цялте два цикъла със съответно $n-1$ и $k \leq n$ брой итерации. Тялото и на двата се изпълнява за константно време.

Тялото на цикъла на ред 14 се изпълнява $k-1$ пъти.

Ред 15 и 16 - конст. време.

Тялото на цикъла на ред 18 се изпълнява $< n$ пъти (за всяко изпълнение на цикъла на р. 14). Редове 19-24 - const.

За цикъла на ред 26 имаме:

- тялото му е за константно време.
- $i_{op} \geq 1$
- $m < n$

Вземам най-голяма сумай: $i_{op} = 1$, $m = n$. Тогава тялото на цикъла на ред 26 се изпълнява n на брой пъти (за всяка итерация на цикъла на ред. 18.

За трите сложни цикъла получавам $k.n.n = k.n^2$.

Следователно сложността по време на алгоритъма Find_min() е $O(k.n^2)$.

Заб. В полето Find_min() получих $O(n^2)$ памет. Можех да я редуцирам до $O(n)$, защото за да изчисля колоната $r+1$, ми трябва стойностите само от колоната r.