

Tiago de Paula Alves

R.A.: 187679

Tiago de Paula Alves

CORRIGIR A QUESTÃO (2)

① VIZINHANÇA-I-ÉSIMO(V, n, i, l)

1. // Acha o i-ésimo menor elemento de V.

2. ~~SELECT~~ SELECT-BFPRT(V, l, n, i)

3. val = V[i]

4.

5. // Monta os pares com a distância (absoluto

6. // da diferença) de cada elemento até o

7. // i-ésimo, junto com o próprio elemento

8. Seja P[1..n] um novo vetor de pares de inteiros.

9. para j=1 até n

10. dist = |V[j] - val|

11. P[j] = (dist, V[j])

12. SELECT-BFPRT(P, 1, n, l)

13.

14. // Copia o resultado para um novo vetor.

15. Seja A[1..l] um novo vetor de inteiros.

16. para j=1 até l

17. (dist, num) = P[j]

18. A[j] = num

19.

20. retorna A

D S T Q O S S
D L M M J V S

O algoritmo acima se baseia no fato de que o SELECT-BFPRT, como apresentado em aula, além de encontrar de encontrar o i -ésimo menor elemento, também particiona o vetor V de forma que elementos menores que o i -ésimo se encontram nas primeiras $i-1$ posições e os maiores nas últimas $n-i$ posições. Isso implica que o i -ésimo elemento vai estar na posição V_i .

Corretude. O primeiro passo do algoritmo é encontrar o i -ésimo menor elemento de V , que após o SELECT-BFPRT se encontra na posição j de V (linhas 2 e 3). Em seguida, é montado um novo vetor P de pares, onde $P_j = (\text{dist}(V_j, V_i), V_j)$ e $\text{dist}(a, b) = |a - b|$ (linhas 8 à 11). Os pares $P_j \in P$ são comparados em ordem lexicográfica, em que o primeiro elemento do par tem maior precedência na comparação.

Após isso, SELECT-BFPRT é chamado com P para encontrar o l -ésimo menor, de acordo com a distância do valor até V_i (linha 12). Devido ao algoritmo de particionamento, temos que todos os pares P_j , com $j \leq l$, compõe as l menores distâncias até o i -ésimo elemento, ou seja, temos ali os l mais próximos do i -ésimo.

A última etapa (linhas 15 a 18) apenas copia os resultados encontrados anteriormente, descartando as distâncias utilizadas para comparação e particionamento.

Complexidade. Como o algoritmo SELECT-BFPRT é $O(n)$, então as linhas 2 até 12 executam em tempo também $O(n)$. ~~Além disso,~~ Além disso, as linhas 15 a 18 executam em $O(l)$. Portanto, o algoritmo como um todo tem complexidade $O(n+l)$.

No entanto, é importante notar que $l \leq n$. Logo, podemos afirmar que o algoritmo executa em tempo $O(n)$, como requerido.

2). Chorãozinho está correto, a divisão em $\lceil \frac{n}{3} \rceil$ grupos é menos eficiente que usando $\lceil \frac{n}{5} \rceil$ grupos no algoritmo BFPRT, para n suficientemente grande.

Dividindo um vetor A de tamanho n em grupos de 3, a recuperação das medianas dos ~~grupos~~ grupos terá complexidade $\Theta(\lceil \frac{n}{3} \rceil) = \Theta(n)$. A partir disso, a mediana das medianas será encontrada em tempo $T(\lceil \frac{n}{3} \rceil)$ e, após o particionamento em $O(n)$, estará na posição k . Para a análise de pior caso, podemos assumir que k não é a solução e a próxima etapa da busca será na maior das partições, com tamanho ~~n_p~~ $n_p = \max\{k-1, n-k\}$ e levando tempo $T(n_p)$.

Note que existem pelo menos $\lceil \frac{1}{2} \lceil \frac{n}{3} \rceil - 1 \rceil$ grupos com medianas menores que o elemento em k , desconsiderando o grupo com menos de 3 elementos, que pode ser o próprio grupo de k . Em cada um desses, teremos garantidos 2 elementos menores que A_k .

Podemos considerar, então, um dos piores casos para a complexidade, em ~~que~~ que k assume seu menor valor possível. Assim:

$$\begin{aligned}\#(A_k) &\leq 2 \left\lceil \frac{1}{2} \left\lceil \frac{n}{3} \right\rceil - 1 \right\rceil + 1 \\ &\leq 2 \left(\frac{1}{2} \left(\frac{n}{3} + 1 \right) - 1 + 1 \right) + 1 \\ &\leq \frac{n}{3} + 2.\end{aligned}$$

Nesse caso, a maior das partições terá $n_p = n - k \geq \frac{2n}{3} - 2 \geq \lfloor \frac{2n}{3} \rfloor - 2$ elementos. A análise considerando o maior valor possível de k chega em uma mesma cota inferior para n_p .

Portanto, considerando que $T(n)$ é crescente, a recorrência do tempo de execução do algoritmo no pior caso será, para ~~$n > 5$~~ ~~algum~~ $a > 0$:

$$T(n) = T(\lceil n/3 \rceil) + T(n_p) + \Theta(n)$$

$$\geq T\left(\left\lceil \frac{n}{3} \right\rceil\right) + T\left(\left\lfloor \frac{2n}{3} \right\rfloor - 2\right) + an \quad \begin{matrix} (\text{quando}) \\ n > 4 \end{matrix}$$

Assumindo $0 < T(1) \leq T(2) \leq t_0$, podemos mostrar que $T(n) \geq cn \lg n$ para algum $c > 0$. Por fim, temos então que, quando o vetor é dividido em grupos de 3 elementos, $T(n) \in \Omega(n \lg n)$, que é assintoticamente ~~mais~~ menos eficiente que o algoritmo original.

Demonstração: Considere a constante $c = \frac{2}{5}$.

Caso base: Seja $3 \leq n \leq 8$. Logo,

$$T(n) \geq T(\lceil n/3 \rceil) + T(n_p) + an \geq an$$

$$\begin{aligned} \text{Mas: } cn \lg n &= \left(\frac{2}{5}\right)n \cancel{\lg n} \lg n \\ &\leq \left(\frac{2}{5}\right)n \lg 8 \\ &= \left(\frac{3}{5}\right)an \\ &\leq T(n) \end{aligned}$$

Hipótese induutiva: Suponha que $T(k) \geq ck \lg k$ para todo $3 \leq k < n$ e algum $n > 8$.

Passo induutivo: Seja $n \geq 9$. Logo,

$$\begin{aligned}
 T(n) &\geq T\left(\left\lceil \frac{n}{3} \right\rceil\right) + T\left(\left\lfloor \frac{2n}{3} \right\rfloor - 2\right) + an \\
 &\geq \left[c \left\lceil \frac{n}{3} \right\rceil \lg \left\lceil \frac{n}{3} \right\rceil \right] + \left[c \left(\left\lfloor \frac{2n}{3} \right\rfloor - 2 \right) \lg \left(\left\lfloor \frac{2n}{3} \right\rfloor - 2 \right) \right] + an \\
 &\geq c \frac{n}{3} \lg \left(\frac{n}{3} \right) + c \left(\frac{2n}{3} - 3 \right) \lg \left(\frac{2n}{3} - 3 \right) + an \\
 &= c \frac{n}{3} \lg \left(\frac{n}{3} \right) + c \left(\frac{2n}{3} - 3 \right) \lg \left(\frac{2n-9}{3} \right) + an \\
 &\geq c \frac{n}{3} \lg \left(\frac{n}{3} \right) + c \left(\frac{2n}{3} - 3 \right) \lg \left(\frac{n}{3} \right) + an \\
 &= c \frac{3n}{3} \lg \left(\frac{n}{3} \right) - 3c \lg \left(\frac{n}{3} \right) + an \\
 &= cn \lg n - cn \lg 3 - 3cn \lg n + 3c \lg 3 + an \\
 &> cn \lg n - cn \lg 3 - 3cn + an \\
 &= cn \lg n + (a - (3 + \lg 3)c)n \\
 &\geq cn \lg n + (a - 5c)n \\
 &= cn \lg n
 \end{aligned}$$

③ A única tarefa possível é a de Chorãozinho, ou seja, é impossível que um algoritmo para este problema tenha complexidade de pior caso $O(n)$.

Note que um algoritmo que resolve o problema dado não tem acesso à capacidade $c(R)$ de um resistor R , podendo apenas saber se $c(R) < c(r)$, $c(R) = c(r)$ ou $c(R) > c(r)$, para um resistor r de cor diferente. Não é possível nem mesmo saber $c(R) - c(r)$ ou $c(R)/c(r)$, que poderia servir como uma medida de distância entre os resistores. Portanto, o algoritmo deve ser baseado puramente em comparações das capacidades.

Supondo então um algoritmo que resolve o problema, ele deve retornar uma ~~per~~ permutação dos resistores amarelos A e outros verdes V , em pares (A_i, V_j) com $0 \leq i, j \leq n$. Logo, esse algoritmo deve ser capaz de escolher entre $(n!)^2$ soluções, das quais $n!$ são corretas, já que a posição dos pares não é importante, bastando que $c(A_i) = c(V_j)$ em cada par (A_i, V_j) da solução.

Podemos ver o algoritmo como uma Árvore de Decisão Ternária, onde cada nó interno é uma comparação de capacidade entre um A_i e um V_j , as arestas são os resultados das comparações e as folhas são as soluções. Portanto, o algoritmo, cuja árvore

tem altura h , deve conter $3^h \geq (n!)^2$ folhas.
Logo, $h \geq \log_3((n!)^2) = 2 \log_3(n!).$

Apesar disso, um algoritmo ótimo poderia organizar as folhas de forma que as $n!$ soluções corretas apareçam em uma mesma subárvore, decidindo entre elas de forma constante. Nesse caso, a subárvore teria altura $h' = \lceil \log_3(n!) \rceil$, e a solução poderia ser encontrada na altura $h^* = h - h' \geq 2 \log_3(n!) - \lceil \log_3(n!) \rceil > \log_3(n!)$, ou seja, $h^* \geq \log_3(n!).$ Entretanto, isso não reduziria a complexidade do algoritmo.

Note que $(n!)^2 > n^n$, portanto:

$$\begin{aligned} h^* &\geq \log_3(n!) \\ &\geq \left(\frac{1}{2}\right) \log_3(n^n) \\ &= \frac{1}{2} \frac{n \lg n}{\lg 3} \\ &= n \lg n / 2 \lg 3 \\ &\geq \frac{1}{4} n \lg n \end{aligned}$$

Com isso, podemos afirmar que o tempo de execução do algoritmo em pior caso será no mínimo $T(n) \geq h^* \geq \frac{1}{4} n \lg n$. Por fim, temos então que $T(n) \in \Omega(n \lg n)$, ou seja, $T(n) \notin O(n)$.

Um algoritmo também poderia ser implementado com base em comparações binárias, representado por uma Árvore de Decisão Binária. Isso não alteraria a complexidade, mas poderia aumentar o número de comparações necessárias.