MAC0338 - Lista 6

Exercício 2

A entrada é uma sequência de números $x_1, x_2, \ldots, x_n$ onde n é par. Projete um algoritmo que particione a entrada em $n/2$ pares da seguinte maneira. Para cada par, computamos a soma de seus números. Denote por $s_1, s_2, \ldots, s_{n/2}$ as $n/2$ somas. O algoritmo deve encontrar uma partição que minimize o máximo das somas e deve ser $O(n \log n)$.

Resposta:

Algorithm 1: MinimizaSomas(x[n])

```
1 Function MINIMIZA-SOMAS(x[n]):
2     ordena(x) ;                                          ▷ O(n log n)
3     cria vetor vazio s com índices de 1 até n/2 ;        ▷ O(n)
4     para i ← 1 até inclusive n/2: ;                      ▷ O(n/2)
5         s_i ← x[i] + x[n − i + 1] ;                      ▷ O(1)
6     return s ;                                           ▷ O(1)
```

Corretude

Vamos provar a corretude do algoritmo acima.

Ordene $x_1 \leq \cdots \leq x_n$.

Caso (x_1, x_n) forme um par, então já temos o que queremos.

Caso contrário, suponha que (a_1, a_k) seja um par tal que $r \neq n$, e (a_n, a_l) seja um par tal que $s \neq 1$. Para fazer a escolha gulosa, troque a_k com a_n e a_l com a_1. Agora temos (a_1, a_n) e (a_k, a_l). Note que

- dentre os pares antigos (a_1, a_k) e (a_n, a_l), $a_1 + a_k \leq a_n + a_l$;
- a soma do novo par $a_1 + a_n$ é menor ou igual a $a_n + a_l$, já que $a_1 \leq a_l$;
- a soma do novo par $a_k + a_l$ é menor ou igual a $a_n + a_l$, já que $a_k \leq a_n$.

Portanto, a escolha gulosa adotada não aumenta o máximo de nenhuma soma.

Agora, considere a sequência sem x_1 nem x_n, ou seja, $x_2, \ldots, x_{n-1}$. Usando o mesmo raciocínio, os pares sempre serão da forma (x_i, x_{n-i+1}) e a soma máxima irá se manter mínima.

Consumo de tempo

Note que, de acordo com as anotações no algoritmo 1, o consumo de tempo é

$$O(n \log n) + O(n) + (O(n/2) \cdot O(1)) + O(1) = O(n \log n),$$

como queríamos.

Exercício 4

Seja $1, \ldots, n$ um conjunto de tarefas. Cada tarefa consome um dia de trabalho; durante um dia de trabalho somente uma das tarefas pode ser executada. Os dias de trabalho são numerados de 1 a n. A cada tarefa t está associado um prazo p_t : a tarefa deveria ser executada em algum dia do intervalo $1..p_t$. A cada tarefa t está associada uma multa $m_t \geq 0$. Se uma dada tarefa t é executada depois do prazo p_t, sou obrigado a pagar a multa m_t (mas a multa não depende do número de dias de atraso).

Problema: Programar as tarefas (ou seja, estabelecer uma bijeção entre as tarefas e os dias de trabalho) de modo a minimizar a multa total. Considere o algoritmo que primeiramente ordena as tarefas pela multa; a seguir, considera uma a uma cada tarefa, escalonando-a no último dia livre dentro do seu prazo. Se não houver dia livre dentro do prazo, escalona a tarefa no último dia livre.

Prove que esse algoritmo está correto, ou seja, produz um escalonamento com multa mínima. Analise seu consumo de tempo.

Resposta:

Seja O uma solução ótima.

Considere uma tarefa t_i dentre as tarefas. Se t_i está escalonada em algum dia antes do próprio prazo, então podemos trocar t_i com a tarefa t_j, escalonada no prazo de t_i, e obter uma solução O' com multa igual a O. Se t_i está escalonada em algum dia depois do próprio prazo, então vamos separar em 2 casos:

- Se $p_{t_i} \leq i$, então existe uma tarefa t_h que possui multa menor ou igual a t_i. Logo, basta trocar t_i com t_h e obter uma solução O' com multa menor que O. Mas como O é ótima, isso não é possível.
- Se $p_{t_i} > i$, então podemos trocar t_i com qualquer tarefa que esteja escalonada após o prazo de t_i e obter uma solução O' com multa menor ou igual a O.

Logo, o algoritmo guloso é ótimo.