Argumentos gulosos comuns

Igor Borja

April 2, 2023

Problema 1. *(Ver `https://cses.fi/problemset/task/1629`) Sejam $(a_1, b_1), \ldots, (a_n, b_n)$ pares de números tais que $a_i < b_i$ para todo $1 \leq i \leq n$. Queremos um algoritmo eficiente para encontrar o **maior** k tal que existem $i_1 \leq \ldots \leq i_k$ com*

$$a_{i_1} \leq b_{i_1} \leq a_{i_2} \leq \ldots \leq a_{i_k} \leq b_{i_k}$$

Solução. Podemos pensar na seguinte analogia: **temos uma lista de processos - cada um com seu tempo de início e término, e queremos determinar qual a maior quantidade de processos que podem ser realizados sem sobreposição.**

Assim, resolveremos utilizando um algoritmo guloso: tomamos sempre, dentre os processos restantes possíveis, aquele que **termina primeiro**.

Teorema 0.1. *Suponha que temos um conjunto de processos R e queremos escolher o maior subconjunto desses processos que pode ter seus elementos executados (em alguma ordem) sem sobreposição, e que o tempo inicial do primeiro processo é maior ou igual a x. Então, devemos escolher como primeiro processo aquele que termina mais cedo.*

Prova. Seja $S(x, U)$ a melhor solução começando do tempo x (isto é, o primeiro processo tem que ter tempo inicial maior ou igual a x) e escolhendo dentre o subconjunto de processos U.

É óbvio que se $U \subseteq V$, $S(x, U) \leq S(x, V)$ pelo simples fato de que a melhor solução para o primeiro caso é válida para o segundo caso. Analogamente, se $y \leq x$, $S(x, U) \leq S(y, U)$. Logo, se $p_j \in R$ é o processo que acaba mais cedo, e escolhemos o processo $p_i \neq p_j \in R$, não podemos escolher p_j em seguida, uma vez que $b_i \geq b_j > a_j$ (começa antes do término de p_i). Segue que a melhor solução que podemos obter é

$$1 + S(b_i, R \setminus \{p_i, p_j\}) \leq 1 + S(b_j, R \setminus \{p_i, p_j\}) \leq 1 + S(b_j, R \setminus p_j)$$

Isso mostra que escolher p_j fornece uma solução pelo menos equivalente e possivelmente melhor. □

□

Problema 2. *(Ver `https://cses.fi/problemset/task/1630`) Sejam $(d_1, f_1), \ldots (d_n, f_n)$ pares de números. Queremos encontrar a permutação $\sigma : \{1, \ldots, n\} \to \{1, \ldots, n\}$ que maximiza*

$$\sum_{1 \leq i \leq n} \left(f_{\sigma(i)} - \sum_{1 \leq j \leq i} d_{\sigma(j)} \right)$$

*Podemos aplicar a seguinte analogia: **cada par $(\mathbf{d_i}, \mathbf{f_i})$ é uma tarefa - com $\mathbf{d_i}$ sendo sua duração e $\mathbf{f_i}$ seu prazo final - e cada tarefa $\mathbf{i}$ é pontudada por $\mathbf{f_i} - \mathbf{f}$, em que $\mathbf{f}$ é o tempo em que ela foi finalizada. Assim, queremos a ordem de processamento das tarefas que maximiza a pontuação**.*

Solução. (Matemática)

Temos que a pontuação é

$$p = \sum_{1 \leq i \leq n} \left(f_{\sigma(i)} - \sum_{1 \leq j \leq i} d_{\sigma(j)} \right) = \sum_{1 \leq i \leq n} f_i - \sum_{1 \leq i \leq n} \sum_{1 \leq j \leq i} d_{\sigma(j)}$$

onde K é uma constante. Porém note que $d_\ell = d_{\sigma(\sigma^{-1}(\ell))}$, e portanto d_ℓ aparece na soma acima para cada $i \geq \sigma^{-1}(\ell)$. Em outras palavras

$$\begin{aligned} p &= K - \sum_{1 \leq i \leq n} (n + 1 - \sigma^{-1}(i)) d_i = K - \sum_{1 \leq i \leq n} (n+1) d_i + \sum_{1 \leq i \leq n} \sigma^{-1}(i) d_i \\ &= K + K' - \sum_{1 \leq i \leq n} i \cdot d_{\sigma(i)} \end{aligned}$$

em que K' é outra constante. Para maximizar p, logo, precisamos minimizar a soma $\sum i \cdot d_{\sigma(i)}$. **Pela desigualdade do rearranjo** torna-se evidente que a permutação escolhida deve ser aquela que satisfaz

$$d_{\sigma(1)} \leq \ldots \leq d_{\sigma(n)}$$

Em outras palavras, devemos escolher as tarefas por ordem crescente de duração, provando o algoritmo guloso. □

Solução. (Algorítimica) Seja σ uma permutação tal que $\sigma(i) > \sigma(i+1)$ para algum $1 \leq i < n$. Mostremos que trocar os dois de posição melhora a pontuação final.

Seja p o preço antes da troca e p' o preço depois da troca. Suponha que a tarefa $T_{\sigma(i)}$ em questão começa em um tempo t. Note que trocar as duas tarefas não interfere no tempo em que as duas vão ter finalizado (que é $t + d_{\sigma(i)} + d_{\sigma(i+1)}$, independente da ordem), e portanto não interfere na pontuação gerada pelas tarefas seguintes - equivalentemente, não interfere nas tarefas anteriores. Seja K a pontuação gerada pelas outras $n-2$ tarefas. Logo

$$\begin{aligned} p' &= K + \big(f_{\sigma(i+1)} - (t + d_{\sigma(i+1)})\big) + \big(f_{\sigma(i)} - (t + d_{\sigma(i+1)} + d_{\sigma(i)})\big) \\ p &= K + \big(f_{\sigma(i)} - (t + d_{\sigma(i)})\big) + \big(f_{\sigma(i+1)} - (t + d_{\sigma(i)} + d_{\sigma(i+1)})\big) \\ p' - p &= \big(-2d_{\sigma(i+1)} - d_{\sigma(i)}\big) - \big(-2d_{\sigma(i)} - d_{\sigma(i+1)}\big) = d_{\sigma(i)} - d_{\sigma(i+1)} > 0 \end{aligned}$$

mostrando que trocar melhora a pontuação final.

Portanto, como desfazer uma inversão sempre melhora a pontuação, é fácil verificar que isso implica que a solução ótima é a permutação σ_0 tal que

$$d_{\sigma_0(1)} \leq \ldots \leq d_{\sigma_0(n)}$$

Ou seja, devemos processar as tarefas em **ordem crescente de distância**, provando o algoritmo guloso. $\square$