# Lista de Exercícios 3

**Pergunta 1.** Seja $G$ um grafo direcionado com pesos positivos nas arestas, seja $s$ um vértice de $G$ e seja $k$ um inteiro positivo. Como podemos encontrar os $k$ vértices mais próximos de $s$? Com qual complexidade? Analise em função de $k, m, n$ ($n$ = número de nós, $m$ = número de arestas).

**Pergunta 2.** Um site de música na Internet decidiu criar um ranking com as melhores canções da última década, a partir de $n$ canções, $s1, \ldots, s_n$, pré-selecionadas. Durante um mês, sempre que um usuário acessava o site, duas canções escolhidas aleatoriamente eram exibidas, e o usuário devia marcar qual das duas ele preferia. Ao término deste processo, temos um conjunto de triplas $S = \{(s_i, s_j, d_{ij} | 1 \leq i < j \leq n\}$, aonde $d_{ij} = i$ se a maioria das pessoas prefere $s_i$ à $s_j$, $d_{ij} = j$ se a maioria das pessoas prefere $s_j$ à $s_i$ e $d_{ij} = 0$ se não há uma preferência entre as canções. Um ranking $R$ é consistente com a lista $S$ se e somente se para todo par de canções $s_i$ e $s_j$ a seguinte condição é válida: se $s_i$ vem antes da canção $s_j$ no ranking $R$, então a maioria das pessoas prefere $s_i$ à $s_j$ ou não há uma preferência entre tais canções.

a) Descreva um algoritmo eficiente para verificar se é possível criar um ranking de canções consistente com a pesquisa. Note que o algoritmo deve responder SIM ou NÃO. Analise a complexidade de pior caso do algoritmo proposto em função de $n$. Explique as estruturas de dados utilizadas para obter tal complexidade.

b) Em algumas situações é possível existir mais de um ranking consistente com $S$. Dado dois rankings $R$ e $R'$ para uma lista $S$, dizemos que $R$ domina $R'$ se e somente se na primeira posição que $R$ difere de $R'$ o índice da canção de $R$ é meno que o índice da canção de $R'$. Por exemplo, se $R = s_1 s_3 s_2 s_4$ e $R' = s_1 s_3 s_4 s_2$, então $R$ domina $R'$ já que os rankings diferem pela primeira vez na terceira posição e $s_2$ tem índice menor que $s_4$. Descreva como seria um algoritmo para determinar um ranking consistente com a lista $S$ que não é dominado por nenhum outro ranking consistente com $S$. Analise a complexidade do algoritmo proposto em função de $n$. Explique as estruturas de dados utilizadas para obter tal complexidade.

**Pergunta 3.** Considere um grafo direcionado acíclico onde cada aresta $e$ tem um tamanho $w_e$. Dado dois vértices $u, v$ no grafo, compute o tamanho do caminho **mais longo** entre eles em tempo $O(n + m)$.

**Pergunta 4.** Um servidor recebe requisições de $n$ clientes no instante 0, uma de cada cliente. Sabemos que o tempo que o servidor leva desde o instante que ele começa a anteder a requisição do cliente $i$ até o instante que ele termina é $t_i$. Além disso, sabemos que o servidor atende apenas uma requisição por vez e quando uma requisição começa a ser atendida ela não pode mais ser interrompida.

a) O tempo de espera de um cliente é definido como o tempo que ele espera até que sua requisição comece a ser atendida. Proponha um algoritmo eficiente para determinar a ordem que o servidor deve atender os clientes de modo a minimizar o tempo médio de espera dos clientes.

**Pergunta 5.** Consider the problem of minimizing the maximum lateness of tasks with **release dates**. There are $n$ tasks to be executed. Each task has a time $t_i$ it takes to be completed, and a due date $d_i$. However, task $i$ is only released at time $r_i$ (so you can only start doing this task after time $r_i$). Assume $d_i \geq r_i + t_i$, so can finish each task **by itself** before the deadline.

As before, the **lateness** of a task if the difference of when it as completed minus its due date. We want to decide when to start working on each task in order to minimize the maximum lateness. Recall at most 1 task can be done at time.

Consider the greedy algorithm that orders the tasks by **increasing due date** and processes them in this order, starting the tasks in the first moment possible (respecting that you can do at most 1 job at a time and the release dates).

Does this greedy algorithm return an optimal schedule (that is, with smallest possible max lateness)? If so, give a brief justification; if not, give a counterexample.

**Pergunta 6.** Given two sorted lists of numbers of length $m, n$. Give an algorithm that finds the $k$th smallest number in the union of the lists, in time $O(\log m + \log n)$. (You can assume that all numbers in the input are distinct).

**Dica:** Em cada iteração você pode descartar a metade de uma das listas. Imagina a posição dos elementos do meio das listas na união ordenada das listas.

**Pergunta 7.** You are given a list of sensitive processes, each has its start and finish times. To *check* a process, you have to do a `status-check` at some point in time while the process is executing (that is, between its start and finish times).

Give an efficient algorithm that, given the start and finish times of all the sensitive processes, finds as small a set of times as possible at which to invoke `status-check`, subject to the requirement that `status-check` is invoked at least once during each sensitive process $P$.

**Dica:** Basta considerar os tempos de término dos processos como possíveis pontos para rodar `status-check`.

**Pergunta 8.** Fomos contratados para definir um planejamento de tarefas para uma equipe para as próximas $n$ semanas. Para cada semana existem 3 tarefas possíveis: $A, B$ e $C$. A realização da tarefa $A$ na semana $i$ gera um lucro de $a_i$; a realização da tarefa $B$ na semana $i$ gera um lucro de $b_i$ e a realização da tarefa $C$ na semana $i$ gera um lucro de $c_i$. Sabe-se também que, para $i = 2, \ldots, n$ a tarefa $C$ só pode ser realizada na semana $i$ se a tarefa $B$ for realizada na semana $i - 1$.

a) Seja $OPT(i)$ o rendimento máximo que pode ser obtido nas $i$ primeiras semanas. Encontre uma equação de recorrência para $OPT(i)$.
b) Escreva um algoritmo polinomial e não recursivo para computar as tarefas que devem ser realizadas a cada semana de modo a maximizar o rendimento total.

**Pergunta 9.** Considere a equação de recorrência que define o número de Lupinutesky: Se $i > 1$ ou $j > 1$ então

$$Lup(i, j) = 3Lup(i - 2, j) + \max_{k=1,\ldots,j-1}\{k + Lup(i, k)\}$$

Se $i \leq 1$ ou $j \leq 1$, então

$$Lup(i, j) = 1$$

a) Desenvolva o PSEUDO-CÓDIGO de um algoritmo polinomial e recursivo para calcular $Lup(n, n)$.
b) Analise a complexidade do algoritmo proposto no item (a) em função de $n$.

**Pergunta 10.** You are going on a long trip. You start on the road at mile post 0. Along the way there are $n$ hotels, at mile posts $a_1 < a_2 < \cdots < a_n$, where each $a_i$ is measured from the starting point. The only places you are allowed to stop are these hotels, but you can choose which of the hotels you stop at. You must stop at the final hotel (at distance $a_n$), which is your destination.

You'd ideally like to travel 200 miles a day, but this may not be possible (depending on the spacing of the hotels). If you travel $x$ miles during a day, the *penalty* for that day is $(200 - x)^2$. You want to plan your trip so as to minimize the total penalty-that is, the sum, over all travel days, of the daily penalties.
Give an efficient algorithm that determines the optimal sequence of hotels at which to stop.

**Pergunta 11.** Dada duas cadeias $X = x_1x_2 \ldots x_n$ e $Y = y_1y_2 \ldots y_m$, queremos encontrar o tamanho da maior subcadeia comum entre $X$ e $Y$, ou seja, o maior valor de $k$ tal que existem índices $i$ e $j$ satisfazendo $x_ix_{i+1} \ldots x_{i+k-1} = y_jy_{j+1} \ldots y_{j+k-1}$. Como exemplo, a maior subcadeia comum entre $X$=ABABD e $Y$=AABABBA é ABAB. Neste caso, $k = 4, i = 1$ e $j = 2$. Seja $X_r$ a subcadeia de $X$ com os $r$ primeiros símbolos de $X$ e $Y_s$ é a subcadeia de $Y$ com os $s$ primeiro símbolos de Y. Por exemplo, se $X$=ABABD, então $X_3$=ABA e $X_2$=AB.

Definimos $OPT(r, s)$ como o **tamanho** da maior subcadeia comum entre $X_r$ e $Y_s$ que inclui $x_r$ e $y_s$. Sabemos que se $x_r = y_s$ e $r, s \geq 1$, então $OPT(r, s) = 1 + OPT(r - 1, s - 1)$. Se $x_r \neq y_s$ ou $r = 0$ ou $s = 0$, então $OPT(r, s) = 0$.

a) Descreva um algoritmo **recursivo e eficiente** para calcular $OPT(n, m)$. Qual a complexidade do algoritmo?
b) Assuma que recebemos uma matriz $M[0..n, 0..m]$ preenchida com os valores de $OPT$, ou seja, $M[i, j] = OPT(i, j)$ para todo $i, j$. Como podemos obter a **maior subcadeia comum** entre $X$ e $Y$ que inclui $x_r$ e $y_s$ a partir da matriz $M$? Note que não está sendo pedido o tamanho da maior subcadeia comum.
c) Assuma que recebemos uma matriz $M[0..n, 0..m]$ preenchida com os valores de $OPT$, ou seja, $M[i, j] = OPT(i, j)$ para todo $i, j$. Como podemos obter o **tamanho** da maior subcadeia comum entre $X$ e $Y$? Qual a complexidade do procedimento?

**Pergunta 12.** A *k-way merge operation*. Suppose you have $k$-sorted arrays, each with $n$ elements, and you want to combine them into a single sorted array of $kn$ elements.

a) Here's one strategy: Using the `merge` procedure from Section 2.3, merge the first two arrays, then merge in the third, then merge in the fourth, and so on. What is the time complexity of this algorithm, in terms of $k$ and $n$?
b) Give a more efficient solution to this problem, using divide-and-conquer.

**Pergunta 13.** O número $B(n)$ de árvores binárias com $n$ nós satisfaz a seguinte equação de recorrência $B(n) = \sum_{i=1}^{n-1} B(i)B(n-i)$ para $n \geq 2$ e $B(1) = 1$.

a) Projete um algoritmo eficiente que recebe $n$ e devolve o número de árvores binárias de tamanho $n$.
b) Analise a complexidade de tempo do algoritmo proposto. (Assuma que toda a multiplicação e adição tem custo constante).

**Pergunta 14.** Seja $A = \{a_1, \ldots, a_n\}$ um conjunto de $n$ números reais positivos. O objetivo desta questão é projetar um algoritmo para encontrar índices $i$ e $j$, com $1 \leq i \leq j \leq n$, tal que $\prod_{k=i}^{j} a_k$ tem o maior valor possível.

a) Como seria um algoritmo de complexidade $O(n^2)$ para resolver este problema?
b) Como seria um algoritmo de complexidade $T(n)$ que satisfaz $\lim_{n\to\infty} \frac{T(n)}{n^2} = 0$?

**Pergunta 15.** An array $A[1..n]$ is said to have a *majority element* if more than half of its entries are the same. Given an array, the task is to design an efficient algorithm to tell whether the array has a majority element, and, if so, to find that element. The elements of the array are not necessarily from some ordered domain like the integers, and so, there can be no comparisons of the form "is $A[i] > A[j]$?". (Think of the array elements as GIF files, say.) However you *can* answer questions of the form "is $A[i] = A[j]$?" in constant time.

a) Show how to solve this problem in $O(n \log n)$ time. (**Hint:** Split the array $A$ into two arrays $A_1$ and $A_2$ of half the size. Does knowing the majority of elements in $A_1$ and $A_2$ help you figure out the majority element of $A$? If so, you can use a divide-and-conquer approach.) [**Hint 2:** Suppose $A$ has a majority element $x$. How many copies of $x$ need to be in $A_1$ or $A_2$? What does this say about the majority of $A_1$ and $A_2$?]

**Pergunta 16.** Give an $O(nt)$ algorithm for the following task.
**Input:** A list of $n$ positive integers $a_1, a_2, \ldots, a_n$; a positive integer $t$.
**Question:** Does some subset of the $a_i$'s add up to $t$? (You can use each $a_i$ at most once.)

**Pergunta 17.** If you select a low-stress job for your team in week $i$, then you get a revenue of $l_i > 0$ dollars; if you select a high-stress job, you get a revenue of $h_i > 0$ dollars. .The catch, however, is that in order for the team to take on a high stress job in week $i$, it's required that they do no job (of either type) in week $i-1$; they need a full week of prep time to get ready for the crushing stress level. On the other hand, it's okay for them to take low-stress job in week $i$ if they have don a job (of either type) in week $i-1$.
**The problem:** Given sets of values $l_1, l_2, \ldots, l_n$ and $h_1, h_2, \ldots, h_n$ find a plan of maximum value. (Such a plan will be called optimal).
**Example:** Suppose $n = 4$, and the values of $l_i$ and $h_i$ are given by the following table.

| | Week 1 | Week 2 | Week 3 | Week 4 |
|---|---|---|---|---|
| $l$ | 10 | 1 | 10 | 10 |
| $h$ | 5 | 50 | 5 | 1 |

Then the plan of maximum value would be to choose ”none” in week 1, a high-stress job in week 2, and low-stress jobs in weeks 3 and 4. The value of this plan would be 0+50+10+10 = 70.

## Exercícios do livro

- Capítulo 4 Kleinberg-Tardos, exercícios: 4, 5, 13.