# Plano de aula de oficina introdutória — Ordenação

Tália de Aguiar
Clube de Programação, UTFPR-CT

13 de novembro de 2025

## Sumário

**1 Estrutura e tempo estimado de aula** **1**

**2 Mark the photographer (cf1705a)** **2**
2.1 Enunciado . . . 2
2.2 Solução . . . 2
2.3 Implementação . . . 2
2.4 Sobre algoritmos gulosos, ou Por que funciona? . . . 2

**3 Tasks and deadlines (cf102961v)** **3**
3.1 Enunciado . . . 3
3.2 Solução . . . 3
3.3 Implementação . . . 3
3.4 Argumento . . . 3

**4 Movie festival (cf102961f)** **4**
4.1 Enunciado . . . 4
4.2 Solução . . . 4
4.3 Implementação . . . 4
4.4 Argumento . . . 4

## 1 Estrutura e tempo estimado de aula

A aula consistirá na discussão de três problemas:

1. Mark the photographer (cf1705a) Tempo: ∼25min
2. Tasks and deadlines (cf102961v) Tempo: ∼10min
3. Movie festival (cf102961f) Tempo: ∼15min

O primeiro tem uma solução que almejamos ser intuitiva para os alunos, mas não tão trivial de se justificar. O andamento esperado é que a solução seja rapidamente sugerida, introduzamos o `sort` da STL somente na medida exigida para a aula, e assim dediquemos uma fração generosa do tempo a argumentar a corretude da solução e a apresentar algoritmos gulosos, ainda que ambos informalmente.

O segundo servirá então como prática simples de um algoritmo guloso.

O terceiro, enfim, será o problema da seleção de atividades, o qual permitirá explorar, além da solução gulosa canônica, proposições gulosas incorretas e seus contraexemplos. Na implementação, introduziremos o `pair` da STL e, tardiamente no curso, `typedef`/`using`, pela conveniência.

# 2 Mark the photographer (cf1705a)

## 2.1 Enunciado

($1 \leq t \leq 100$ casos de teste.) Mark tirará uma foto de $2n$ pessoas ($1 \leq n \leq 100$). A $i$-ésima tem altura $h_i$. Ele deverá organizá-las em uma fileira da frente e uma de trás, ambas com $n$ pessoas, mas de modo que cada pessoa que fique atrás seja pelo menos $x$ unidades mais alta que a à sua frente. Isso é possível?

## 2.2 Solução

Ordenamos as pessoas por altura, e pareamos a $i$-ésima ($1 \leq i \leq n$) com a $(n+i)$-ésima. Podemos atender à condição pedida se e somente se essa configuração lhe atende.

## 2.3 Implementação

Não apresentaremos algoritmos de ordenação em si (estudados em Estrutura de Dados 1), pois, predominantemente, as soluções em Programação Esportiva empregam ordenação como mera subrotina, sem agir nos detalhes de sua implementação. Usaremos a função `sort` da STL. Para nossas finalidades, basta sabermos que ela ordena os elementos de um vetor de forma não-decrescente, e que sua complexidade é de $O(n \log n)$ comparações.

```
sort(h.begin(), h.end());
bool res = true;
for(int i = 0; i < n && res; ++i)
    if(h[n+i] < h[i]+x) res = false;
if(res) {
    cout << "YES\n";
} else {
    cout << "NO\n";
}
```

## 2.4 Sobre algoritmos gulosos, ou Por que funciona?

Apresentaremos algoritmos gulosos um tanto informalmente. Isso porque, não raro, chegamos a uma solução gulosa por intuição e nos limitamos a esboçar um argumento em seu favor, uma vez que estruturá-la e demonstrá-la formalmente demandaria muito mais tempo. Assim, no contexto do esporte, o termo acaba sendo empregado leve e sugestivamente.

Dada a ressalva, o que queremos dizer em geral com algoritmos gulosos? Alguns problemas pedem que realizemos uma série de escolhas, cada uma podendo afetar as futuras. No problema visto, por exemplo, devemos escolher como dividir as pessoas em duas fileiras, e então parear cada uma à frente com uma atrás. Ao dividi-las, delimitamos quais pares podemos formar, e, ainda, a cada par, restringimos mais as opções dos próximos. Às vezes, porém, conhecemos uma escolha ótima no momento e que não gerará prejuízo para as futuras, mesmo sem explorar totalmente suas ramificações. Essa é a chamada escolha gulosa. Um algoritmo guloso é apenas um algoritmo que sempre toma decisões gulosas.

Pôr as menores pessoas à frente, e parear a $i$-ésima menor ($1 \leq i \leq n$) sempre com a $(n+i)$-ésima são ambos exemplos de escolhas gulosas. Que argumentação podemos esboçar em favor delas? Examinemos primeiro a divisão das fileiras. Admitamos, por hipótese, que haja uma configuração que atenda à condição pedida e em que uma $i$-ésima ($1 \leq i \leq n$) menor pessoa fique atrás. Então necessariamente uma $j$-ésima ($n+1 \leq j \leq 2n$) estará à frente. Nesse caso, podemos trocar $i$ e $j$ de lugar, pois sendo a pessoa à frente de $i$ baixa o bastante para ela, será também para $j$, que é mais alta, e, analogamente, a pessoa atrás de $j$ será alta para $i$. Repetindo essa operação, concluímos que, se há uma configuração válida sem que as menores pessoas estejam à frente, há também uma em que elas estão, tal que podemos escolher fazê-lo assim sem prejuízo.

Justifiquemos agora o pareamento. Admitamos, por hipótese, que haja uma configuração com as menores pessoas à frente e que atenda à condição pedida, mas em que haja dois pares $(i_1, j_1), (i_2, j_2)$ ($1 \leq i_1, i_2 \leq n$, $n+1 \leq j_1, j_2 \leq 2n$) tais que $i_1 < i_2$ e $j_1 > j_2$. Similarmente à argumentação anterior, podemos trocar $j_1$ e $j_2$ continuando com uma configuração válida. Repetindo essa operação, concluímos que podemos escolher ordenar a fileira traseira conforme a ordenação da frontal sem prejuízo.

# 3 Tasks and deadlines (cf102961v)

## 3.1 Enunciado

Começando no instante 0, atenderemos a $n$ tarefas ($1 \leq n \leq 2 \cdot 10^5$) em ordem de nossa escolha. A $i$-ésima tem duração $a_i$ e prazo $d_i$. Seja $f_i$ o instante em que a terminaremos. Nossa recompensa por ela será $d_i - f_i$ (podendo ser negativa!), e nossa recompensa total será a soma das recompensas de cada tarefa. Qual é a máxima recompensa total possível?

## 3.2 Solução

Observemos que a recompensa total será a soma dos $d$, que é sempre a mesma, menos a soma dos $f$, que é, portanto, o que desejamos minimizar. Para isso, escolhemos gulosamente atender às tarefas da mais curta para a mais longa.

## 3.3 Implementação

```
ll sum_d = 0;
for(int i = 0; i < n; ++i) sum_d += d[i];
sort(a.begin(), a.end());
ll f = 0, sum_f = 0;
for(int i = 0; i < n; ++i) {
    f += a[i];
    sum_f += f;
}
cout << sum_d-sum_f << '\n';
```

## 3.4 Argumento

O $f$ de cada tarefa será a soma dos $a$ dela e das tarefas anteriores. Uma perspectiva mais frutífera, porém, é a de que o $a$ de cada tarefa contribuirá em seu $f$ e no de todas as tarefas posteriores. Assim, é intuitivo que deixemos tarefas mais longas para o fim, de modo a contabilizá-las menos

vezes, e, reciprocamente, que as tarefas iniciais, cujo $a$ será contabilizado mais vezes, sejam as mais curtas.

# 4 Movie festival (cf102961f)

## 4.1 Enunciado

Em um festival, serão exibidos $n$ filmes ($1 \leq n \leq 2 \cdot 10^5$). O $i$-ésimo terá início e fim $[a_i, b_i[$ ($1 \leq a_i < b_i \leq 10^9$). Qual é o máximo de filmes a que poderíamos assistir inteiramente?

## 4.2 Solução

Este é um problema clássico, que podemos encontrar como *o problema da seleção de atividades*. Algumas proposições gulosas que não funcionam são escolher sempre o filme mais curto, se possível, ou sempre o de menor $a$, se possível. Uma solução gulosa canônica, porém, é escolher sempre o de menor $b$, se possível (ou, analogamente, sempre o de maior $a$).

## 4.3 Implementação

Como faríamos para ordenar os filmes por $b$? Se ordenássemos somente os $b$, perderíamos os $a$ correspondentes. Uma opção seria vincularmos os pares $a$ e $b$ usando a estrutura `pair` da STL, a qual representa um par ordenado, cujos elementos podem ser inclusive de tipos diferentes. Um par de dois inteiros seria `pair<int, int>`. Acessamos o primeiro elemento de um par $p$ por `p.first` e o segundo por `p.second`. Ao ordenarmos pares, ordenamos conforme o primeiro elemento, ou, em caso de empate, conforme o segundo. (Esse critério é chamado ordenação lexicográfica, referindo-se a como ordenamos palavras.) Portanto, guardaremos os filmes como pares $(b, a)$, ainda que o desempate por $a$ seja supérfluo nesta solução.

```
vector<ii> p(n);
for(int i = 0; i < n; ++i) cin >> p[i].second >> p[i].first;
sort(p.begin(), p.end());
int res = 0, last_b = 0;
for(int i = 0; i < n; ++i) {
    int a = p[i].second, b = p[i].first;
    if(last_b <= a) {
        ++res;
        last_b = b;
    }
}
cout << res << '\n';
```

## 4.4 Argumento

Examinemos primeiro a escolha gulosa de incluir o filme de menor $b$. Admitamos, por hipótese, que haja uma seleção máxima de filmes que não o inclua, e consideremos o filme de menor $b$ nessa seleção. Podemos trocá-lo pela escolha gulosa sem conflitos, mostrando que ela não gera prejuízo para o máximo final, mesmo quando exclui alguns filmes no momento. Repetimos então esse processo para os filmes restantes, sempre com a mesma argumentação.