Construção e Análise de Algoritmos

aula 08: Divisão e conquista

1 Introdução

A ideia é

- *Porque fazer a coisa toda de uma vez*
  se a gente pode fazer primeiro a metade
  e depois a outra metade ?

Mas não é só isso, claro.

Quer dizer, a ideia é que as metades também vão ser resolvidas assim — (*a coisa é recursiva ...*)

Nem sempre a coisa é dividida em metades — (*nós podemos ter outros tamanhos, e mais do que dois subproblemas*)

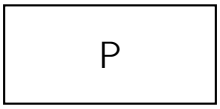
E em geral, é preciso fazer mais alguma coisa antes ou depois de resolver os subproblemas.

Essa estratégia de raciocínio é conhecida como a *Divisão e Conquista.*

Vejamos como a coisa funciona.

2 Divisão e conquista

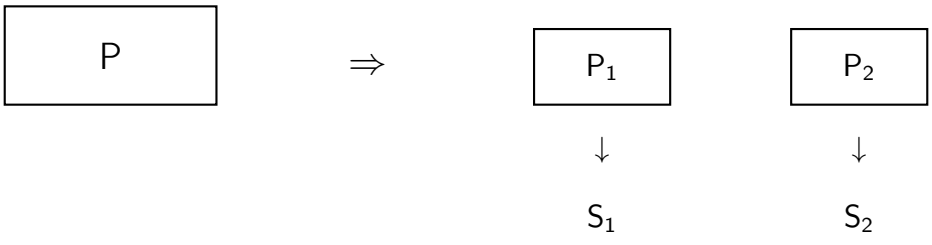
Imagine que nós temos um problema P



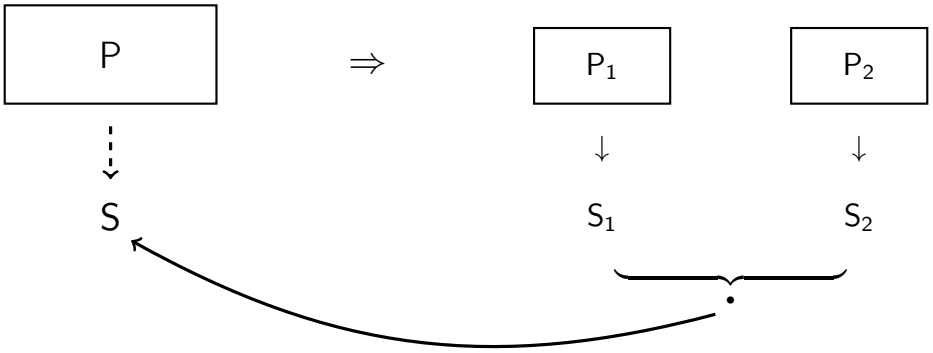
Imagine que nós conseguimos dividir o problema P em dois

— (*do mesmo tipo que* P)

Imagine que nós temos as soluções de P_1 e P_2



E imagine que nós sabemos como combinar as soluções S_1 e S_2 para obter a solução do problema P



Então agora acabou.

Quer dizer, nós já sabemos como resolver o problema P.

Mas, como assim?

Eu só estou imaginando que eu sei resolver $\mathsf{P_1}$ *e* $\mathsf{P_2}$ — *(quando na prática eu não sei ...)*

E além do mais, $\mathsf{P_1}$ *e* $\mathsf{P_2}$ *são do mesmo tipo que* P.

Então na prática, eu estou assumindo que eu sei fazer aquilo que eu não sei fazer.

Isso não pode estar certo ...

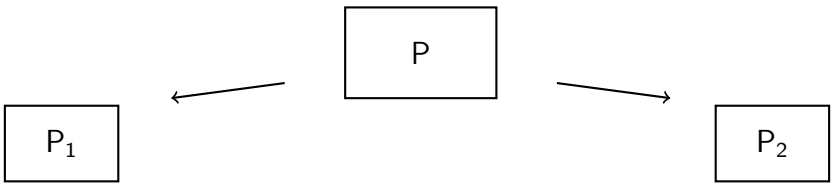
Pois é ...

Mas o pior é que está.

Então explique melhor.

Tá bom.

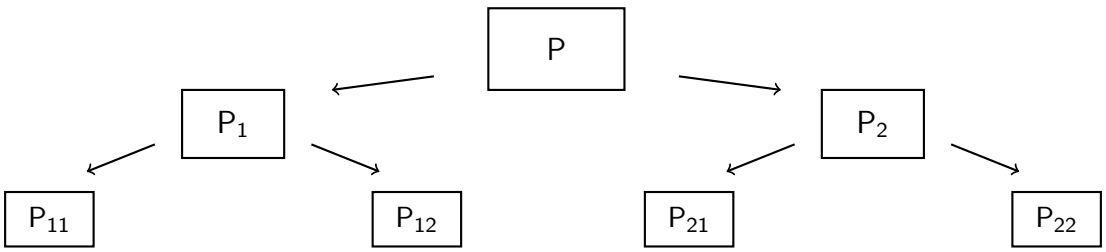
Mas para isso, é melhor organizar a coisa assim



Agora, lembre que $\mathsf{P_1}$ e $\mathsf{P_2}$ são do mesmo tipo que P.

E lembre que a gente já sabe dividir o problema P.

Então, a gente também sabe dividir $\mathsf{P_1}$ e $\mathsf{P_2}$



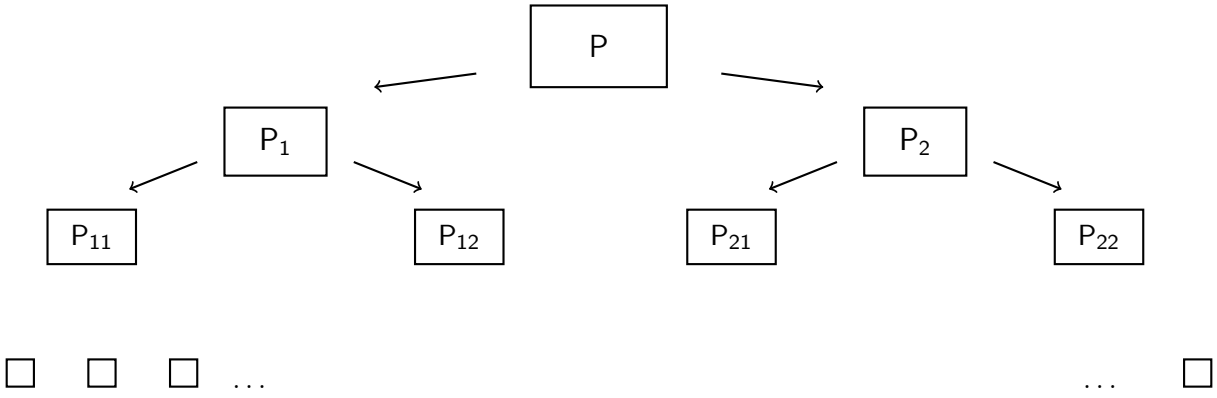
— (*onde* $\mathsf{P_{11}}$, $\mathsf{P_{12}}$, $\mathsf{P_{21}}$, $\mathsf{P_{22}}$ *são do mesmo tipo que* P)

Entendi ...

Então agora, a gente pode continuar dividindo os subproblemas.

É isso aí ...

Dividindo a coisa pra sempre, a gente chega em probleminhas de tamanho 1



E a ideia é que a gente sempre sabe resolver um probleminha de tamanho 1.

Entendi ...

E dái, quando a gente tiver as soluções lá embaixo, a gente vai conseguir subir.

Porque a gente sabe como combinar as soluções.

É isso aí ...

2.1 O algoritmo de divisão e conquista

Agora, o mais legal é que a gente consegue escrever essa ideia no formato de pseudo-código

```
func Algoritmo-DC ( P )
    Se (P é pequenininho)
        Retorna (Algoritmo-base(P))
    P1,P2 ← Quebra (P)
    S1 ← Algoritmo-DC (P1)
    S2 ← Algoritmo-DC (P2)
    S ← Combina (S1,S2)
    Retorna (S)
```

Entendi ...

Então é por isso que todos os nossos algoritmos tem a mesma cara.

É isso aí ...

E agora, é fácil ver outros exemplos de algoritmos de divisão e conquista.

Exemplos

a. Mergesort

Nós vimos na aula 03 o primeiro exemplo de algoritmo de divisão e conquista

```
func Mergesort (L, inicio, fim)
    Se (inicio = fim)
        Retorna
    meio ← (inicio + fim) / 2
    Mergesort (L, inicio, meio)
    Mergesort (L, meio+1, fim)
    Intercala (L, inicio, meio, fim)
```

◇

b. Busca binária

E na disciplina de Estruturas de Dados, nós também encontramos um algoritmo de divisão e conquista

```
func BB (k, L, inicio, fim)
    Se (inicio > fim)          Retorna (0)
    meio ← (inicio + fim) / 2
    Se (L[meio] = k)           Retorna (1)
    SSe (L[meio] > k)          Retorna (BB (k, L, inicio, meio-1))
    Senão                      Retorna (BB (k, L, meio+1, fim))
```

◇

3 O método das equações de recorrência

Depois que a gente escreve um algoritmo de divisão e conquista, a gente pode obter o seu tempo de execução por meio da anotação do código.

Para isso, a gente faz a anotação do código assim

```
T(n) =   func Mergesort-DC (L, inicio, fim)
          |  Se (inicio = fim)
  O(1)    |      Retorna
          |  meio ← (inicio + fim) / 2
T(n/2) ←  Mergesort (L, inicio, meio)
T(n/2) ←  Mergesort (L, meio+1, fim)
  O(n)    |  Intercala (L, inicio, meio, fim)
```

Quer dizer, a ideia é que $T(n)$ é a função que especifica o tempo de execução do algoritmo.

E as anotações estão dizendo que

$$T(n) \quad = \quad 2 \cdot T\,(n/2) \quad + \quad O(n)$$

Sim! uma equação de recorrência — (*ou um rocambole ...*)

E para obter a sua solução, basta desenrolar o rocambole

$$\begin{aligned}
T(n) &= 2 \cdot T\left(\tfrac{n}{2}\right) + n \\
&= 2 \cdot \left[2 \cdot T\left(\tfrac{n}{2^2}\right) + \tfrac{n}{2}\right] + n \\
&= 2^2 \cdot T\left(\tfrac{n}{2^2}\right) + n + n \\
&= 2^3 \cdot T\left(\tfrac{n}{2^3}\right) + n + n + n \\
&= \ldots \\
&= 2^{\log n} \cdot T\left(\tfrac{n}{2^{\log n}}\right) + \underbrace{n + \ldots + n}_{\log n} \\
&= n \cdot \cancel{T(1)} + n + \ldots + n \\
&= O(n \log n)
\end{aligned}$$

Vejamos mais exemplos.

Exemplos

a. Busca binária

Abaixo nós temos o código da busca binária já anotado

```
T(n) =  func BB ( k, L, inicio, fim )
          Se ( inicio > fim )          Retorna ( 0 )
O(1)      meio ← ( inicio + fim ) / 2
          Se ( L[meio] = k )           Retorna ( 1 )
T(n/2) ←  SSe ( L[meio] > k )          Retorna ( BB ( k, L, inicio, meio-1 ) )
T(n/2) ←  Senão                        Retorna ( BB ( k, L, meio+1, fim ) )
```

E aqui, a esperteza é notar que no máximo uma chamada recursiva é realizada.

Logo, a equação de recorrência fica assim

$$T(n) \quad = \quad T\left(\frac{n}{2}\right) \quad + \quad O(1)$$

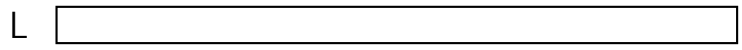
E agora, é só desenrolar o rocambole

$$
\begin{aligned}
T(n) &= T\left(\tfrac{n}{2}\right) + 1 \\
&= T\left(\tfrac{n}{2^2}\right) + 1 + 1 \\
&= T\left(\tfrac{n}{2^3}\right) + 1 + 1 + 1 \\
&= \dots \\
&= T\left(\tfrac{n}{2^{\log n}}\right) + \underbrace{1 + \dots + 1}_{\log\, n} \\
&= O(\log n)
\end{aligned}
$$

◇

b. Ordenação 2/3

Imagine que L é uma lista de tamanho n que contém números inteiros

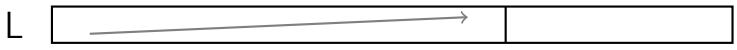


E imagine que o problema é

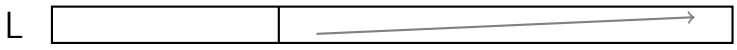
→ *Colocar a lista* L *em ordem crescente*

Daí, uma ideia é fazer o seguinte

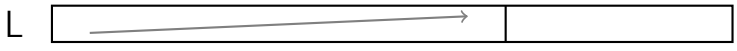
1. Ordenar os primeiros $2n/3$ elementos



2. Depois ordenar os últimos $2n/3$ elementos



3. E depois ordenar os primeiros $2n/3$ elementos outra vez



— (*onde cada ordenação é feita recursivamente pelo mesmo método*)

Ao final do processo, a lista está completamente ordenada — (*porque?*)

E isso nos dá mais um algoritmo de divisão e conquista.

Mas, será que isso é uma boa ideia?

Bom, para descobrir isso nós vamos analisar o tempo de execução do algoritmo.

E para isso, nós escrevemos e anotamos o seu pseudo-código

```
T(n) =  func Ordenação-2/3 ( L, inicio, fim )
                Se ( inicio ≥ fim )          Retorna
        O(1)    T1 ← ( 2*inicio + fim ) / 3
                T2 ← ( inicio + 2*fim ) / 3
T(2n/3) ←       Ordenação-2/3 ( L, inicio, T2 )
T(2n/3) ←       Ordenação-2/3 ( L, T1+1, fim )
T(2n/3) ←       Ordenação-2/3 ( L, inicio, T2 )
```

Com base nas anotações a gente obtém a seguinte equação de recorrência

$$T(n) \quad = \quad 3 \cdot T\left(\frac{2n}{3}\right) \quad + \quad O(1)$$

E agora, é só desenrolar o rocambole

$$\begin{aligned}
T(n) \quad &= \quad 3 \cdot T\left(\tfrac{2n}{3}\right) \quad + \quad 1 \\
&= \quad 3 \cdot \left[3 \cdot T\left(\tfrac{n}{(3/2)^2}\right) \quad + \quad 1 \right] \quad + \quad 1 \\
&= \quad 3^2 \cdot T\left(\tfrac{n}{(3/2)^2}\right) \quad + \quad 3 \quad + \quad 1 \\
&= \quad 3^3 \cdot T\left(\tfrac{n}{(3/2)^3}\right) \quad + \quad 3^2 \quad + \quad 3 \quad + \quad 1 \\
&= \quad \ldots \\
&= \quad 3^{\log_{3/2}(n)} \cdot \cancel{T(1)} \quad + \quad 3^{\log_{3/2}(n)-1} \quad + \quad \ldots \quad + \quad 3 \quad + \quad 1
\end{aligned}$$

Quer dizer, nós chegamos a

$$T(n) \quad = \quad 1 \quad + \quad 3 \quad + \quad 3^2 \quad + \quad \ldots \quad + \quad 3^{\log_{3/2}(n)}$$

— (*uma soma de PG ...*)

Então agora, a gente usa o truque da soma de PG

$$S \quad = \quad 1 \quad + \quad 3 \quad + \quad 3^2 \quad + \quad \ldots \quad + \quad 3^{\log_{3/2}(n)}$$

Quer dizer, o truque é colocar o 3 em evidência do lado direito

$$S \quad = \quad 1 \quad + \quad 3 \cdot \left[1 \quad + \quad 3 \quad + \quad \ldots \quad + \quad 3^{\log_{3/2}(n)-1} \right]$$

E ver que o S apareceu dentro dos colchetes — (*ou quase isso ...*)

$$S \quad = \quad 1 \quad + \quad 3 \cdot \left[S \quad - \quad 3^{\log_{3/2}(n)} \right]$$

O que nos dá

$$S \quad = \quad \frac{3^{\log_{3/2}(n)+1} \quad - \quad 1}{2} \quad = \quad \frac{3}{2} \cdot 3^{\log_{3/2}(n)} \quad - \quad \frac{1}{2}$$

Nesse ponto, a gente olha para o termo

$$3^{\log_{3/2}(n)}$$

E nota que o 3 pode ser escrito assim

$$3 \quad = \quad \big(3/2\big)^{\log_{3/2}(3)}$$

Então, a gente pode reescrever o nosso termo assim

$$\Big(\big(3/2\big)^{\log_{3/2}(3)} \Big)^{\log_{3/2}(n)}$$

e depois assim

$$\big(3/2\big)^{\log_{3/2}(3) \cdot \log_{3/2}(n)}$$

e depois assim

$$\Big(\big(3/2\big)^{\log_{3/2}(n)} \Big)^{\log_{3/2}(3)}$$

e depois assim

$$n^{\log_{3/2}(3)}$$

e depois assim

$$n^{2,7}$$

E fazendo isso, a gente descobre que o nosso algoritmo é uma grande porcaria.

◇

4 Análise de caso médio do Quicksort

Abaixo nós temos o pseudo-código do algoritmo Quicksort

```
func Quicksort ( L, inicio, fim )
    Se ( inicio ≥ fim )
        Retorna
    k ← Partição ( L, inicio, fim )
    Quicksort ( L, inicio, k )
    Quicksort ( L, k+1, fim )
```

Só que dessa vez, não é possível anotá-lo.

Porque a gente não sabe o tamanho das partes produzidas pela função Partição().

Quer dizer, no pior caso uma das partes fica vazia, e a outra tem $n - 1$ elementos.

Então, a gente pode anotar o código assim

```
T(n) =  func Quicksort ( L, inicio, fim )
          Se ( inicio ≥ fim )
O(1)        Retorna
O(n)      k ← Partição ( L, inicio, fim )
T(n-1) ←  Quicksort ( L, inicio, k )
T(0) ←    Quicksort ( L, k+1, fim )
```

E nós obtemos a seguinte equação de recorrência

$$T(n) \quad = \quad T(n-1) \quad + \quad O(n)$$

Agora, é só desenrolar o rocambole

$$\begin{aligned}
T(n) &= T(n-1) + n \\
&= T(n-2) + (n-1) + n \\
&= T(n-3) + (n-2) + (n-1) + n \\
&= \ldots \\
&= T(1) + 2 + 3 + \ldots + (n-1) + n \\
&= 1 + 2 + 3 + \ldots + (n-1) + n \\
&= O(n^2)
\end{aligned}$$

O problema é que a análise de pior caso não é muito informativa nesse caso.

Porque na prática o pior caso não acontece.

Então o que nós precisamos é de uma análise de caso médio.

Mas, o que é o caso médio?

Bom, ele é a média aritmética de todos os casos.

E quais são todos os casos?

Bom, existe o pior caso

E existe o segundo pior caso

E existe o terceiro pior caso

Quer dizer, são dois casos de cada tipo.

E em todos eles, nós temos duas chamadas recursivas.

Então, quando a gente soma tudo dá

$$2 \cdot T(0) \quad + \quad 2 \cdot T(1) \quad + \quad \ldots \quad + \quad 2 \cdot T(n-2) \quad + \quad 2 \cdot T(n-1)$$

E a média aritmética fica assim

$$\frac{2 \cdot T(0) \quad + \quad 2 \cdot T(1) \quad + \quad \ldots \quad + \quad 2 \cdot T(n-2) \quad + \quad 2 \cdot T(n-1)}{n}$$

Agora, a gente pode escrever a equação de recorrência assim

$$T(n) \quad = \quad \frac{2 \cdot T(0) \quad + \quad 2 \cdot T(1) \quad + \quad \ldots \quad + \quad 2 \cdot T(n-2) \quad + \quad 2 \cdot T(n-1)}{n} \quad + \quad n$$

Só que isso não é um rocambole que a gente consegue desenrolar.

Então agora é a hora para uma esperteza.

Quer dizer, a esperteza é tentar ver o caso $n-1$ dentro do caso n

E para isso, a gente foca a atenção nessa parte aqui

$$T(n) \quad = \quad \underbrace{\frac{2 \cdot T(0) \quad + \quad 2 \cdot T(1) \quad + \quad \ldots \quad + \quad 2 \cdot T(n-2) \quad + \quad 2 \cdot T(n-1)}{n}}_{*} \quad + \quad n$$

Daí, o primeiro passo é separar o último termo

$$T(n) \quad = \quad \frac{2 \cdot T(0) \quad + \quad 2 \cdot T(1) \quad + \quad \ldots \quad + \quad 2 \cdot T(n-2)}{n} \quad + \quad \frac{2 \cdot T(n-1)}{n} \quad + \quad n$$

Depois, multiplicar e dividir por $n-1$

$$T(n) \quad = \quad \frac{2 \cdot T(0) \quad + \quad 2 \cdot T(1) \quad + \quad \ldots \quad + \quad 2 \cdot T(n-2)}{n} \cdot \frac{n-1}{n-1} \quad + \quad \frac{2 \cdot T(n-1)}{n} \quad + \quad n$$

E depois trocar os denominadores, para ver o seguinte

$$T(n) \quad = \quad \underbrace{\frac{2 \cdot T(0) \quad + \quad 2 \cdot T(1) \quad + \quad \ldots \quad + \quad 2 \cdot T(n-2)}{n-1}}_{T(n-1) \quad - \quad (n-1)} \cdot \frac{n-1}{n} \quad + \quad \frac{2 \cdot T(n-1)}{n} \quad + \quad n$$

Agora, a gente pode reescrever a recorrência assim

$$T(n) \quad = \quad \Big[\, T(n-1) \;-\; (n-1) \,\Big] \cdot \frac{n-1}{n} \quad + \quad \frac{2 \cdot T(n-1)}{n} \quad + \quad n$$

depois reescrever assim

$$T(n) \quad = \quad T(n-1) \cdot \frac{n+1}{n} \quad - \quad \frac{(n-1)^2}{n} \quad + \quad n$$

e depois assim

$$T(n) \quad = \quad T(n-1) \cdot \frac{n+1}{n} \quad + \quad 2 \quad - \quad \frac{1}{n}$$

e depois assim

$$T(n) \quad = \quad T(n-1) \cdot \frac{n+1}{n} \quad + \quad 2$$

— (*jogando fora um termo que não vai fazer diferença, e reduzindo a constante ...*)

Agora a gente tem um rocambole que dá para desenrolar.

Então a gente vai fazer isso

$$\begin{aligned}
T(n) \quad &= \quad T(n-1) \cdot \tfrac{n+1}{n} \quad + \quad 1 \\
&= \quad \Big[\, T(n-2) \cdot \tfrac{n}{n-1} \quad + \quad 1 \,\Big] \cdot \tfrac{n+1}{n} \quad + \quad 1 \\
&= \quad T(n-2) \cdot \tfrac{n+1}{n-1} \quad + \quad \tfrac{n+1}{n} \quad + \quad 1 \\
&= \quad T(n-3) \cdot \tfrac{n+1}{n-2} \quad + \quad \tfrac{n+1}{n-1} \quad + \quad \tfrac{n+1}{n} \quad + \quad 1 \\
&= \quad \ldots \\
&= \quad T(0) \cdot \tfrac{n+1}{1} \quad + \quad \tfrac{n+1}{2} \quad + \quad \ldots \quad + \quad \tfrac{n+1}{n-1} \quad + \quad \tfrac{n+1}{n} \quad + \quad 1
\end{aligned}$$

Daí, colocando o $n+1$ em evidência a gente obtém o seguinte

$$T(n) \quad = \quad 1 \quad + \quad (n+1) \cdot \Big[\, 1 \quad + \quad \frac{1}{2} \quad + \quad \frac{1}{3} \quad + \quad \ldots \quad + \quad \frac{1}{n-1} \quad + \quad \frac{1}{n} \,\Big]$$

Aqui, a gente reconhece a soma harmônica dentro dos colchetes.

E todo mundo sabe que essa soma vale (aprox.) $\log n$ — (*ou será que não?*)

Então agora, a gente pode concluir que

$$T(n) \quad = \quad O(n \log n)$$

Legal, não é?