Projeto de algoritmos recursivos. Divisão e conquista

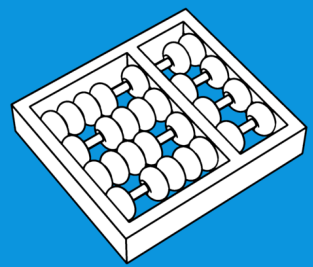
MO417 - Complexidade de Algoritmos I

04/24

8

Santiago Valdés Ravelo
https://ic.unicamp.br/~santiago/
ravelo@unicamp.br

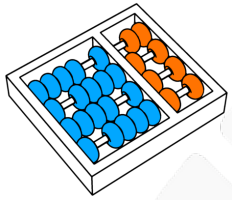
UNICAMP



“Divide et impera”

Júlio César.

Última aula



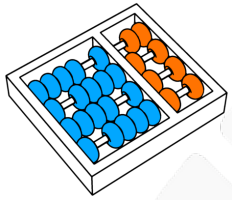
Projeto de algoritmos recursivos

Uma estratégia é:

1. Encontrar e definir um **SUBPROBLEMA** adequado.
2. Supor que sabemos resolver instâncias menores.
3. Construir um algoritmo recursivo para o subproblema.

Este processo é análogo a **DEMONSTRAÇÕES POR INDUÇÃO**:

- **Caso básico:** instâncias pequenas.
- **Caso geral:** instâncias grandes:
  - A chamada recursiva corresponde à **HIPÓTESE DE INDUÇÃO**.
  - A construção da solução corresponde ao **PASSO DA INDUÇÃO**.

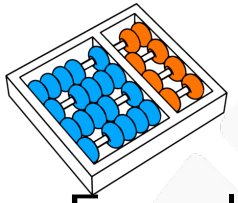


Celebridades

Definição

*Em um conjunto S de n pessoas, uma **celebridade** é alguém que é conhecido por todas as pessoas de S mas que não conhece ninguém.*

- Nem sempre existe uma celebridade.
- Só pode existir uma celebridade em S.
- Queremos saber se existe uma celebridade em S.



O problema da celebridade

Formalizando o problema:

- Queremos descrever o conhecimento entre n pessoas.
- Representamos isso com uma matriz binária $n \times n$.
- $M[i, j] = 1$ se i conhece j, e $M[i, j] = 0$ caso contrário.

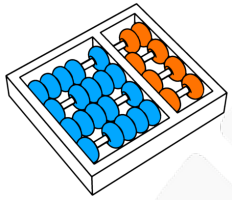
Problema

Entrada: *Um conjunto de n pessoas e a matriz associada M.*

Saída: *Uma celebridade, se existir.*

Observações:

- Para uma celebridade k, vale $M[i, k] = 1$ para todo i.
- Analogamente, vale $M[k, i] = 0$ para todo i.
- Um algoritmo simples verifica se cada pessoa é celebridade, usando $2n(n - 1)$ operações.



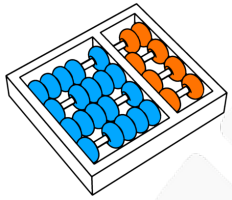
Argumento indutivo

Um argumento indutivo pode ser:

Teorema (Hipótese de indução)

Sabemos encontrar uma celebridade em um conjunto de $n-1$ pessoas ou decidir que não existe.

- Se $n = 1$, a única pessoa no conjunto é uma celebridade.
- Se $n \geq 2$:
  - Represente o conjunto de pessoas como $S = \{1, 2, \ldots, n\}$.
  - Podemos encontrar celebridade em $S' = \{1, 2, \ldots, n-1\}$.
  - Como isso ajuda na instância original?

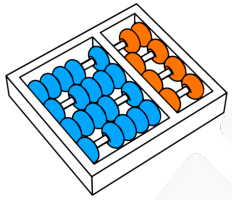


Passo indutivo

Dois casos:

1. Existe celebridade k em S':
   - Se $M[n,k] = 1$ e $M[k,n] = 0$, então k é celebridade em S.
   - Senão, n ainda poderia ser celebridade.
2. **NÃO** existe celebridade em S':
   - Nesse caso, apenas n poderia ser celebridade.

- Em qualquer caso, precisamos verificar se n é celebridade.
- Temos que verificar se $M[n,j] = 0$ e $M[j,n] = 1$ para $j < n$.
- Também obtemos um algoritmo quadrático.



Segunda tentativa

Suponha que s não é celebridade

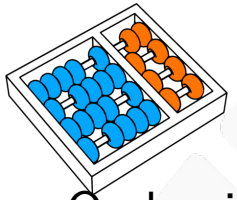
- Podemos fazer a chamada recursiva para $S' = S \setminus \{s\}$.
- Não precisaríamos verificar s depois.

Como encontrar alguém que **NÃO** é celebridade?

- Considere duas pessoas i e j.
- Se $M[i, j] = 1$, então i não é celebridade.
- Se $M[i, j] = 0$, então j não é celebridade.

Vamos usar a mesma hipótese anterior:

- Consideramos o mesmo subproblema.
- Escolhemos a subinstância mais cuidadosamente!

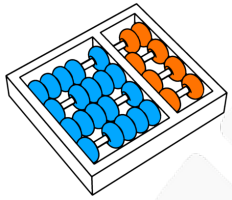


Encontrando uma celebridade

O algoritmo devolve NIL se não houver celebridade.

Algoritmo: CELEBRIDADE(S, M)

```
1  se |S| = 1
2      seja k a única pessoa em S
3  senão
4      sejam i, j duas pessoas em S
5      se M[i, j] = 1
6          s ← i
7      senão
8          s ← j
9      S' ← S \ {s}
10     k ← CELEBRIDADE(S', M)
11     se k ≠ NIL e (M[s, k] = 0 ou M[k, s] = 1)
12         k ← NIL
13 devolva k
```



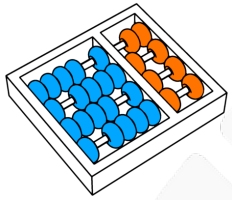
Exemplo 4 - Complexidade

Seja $T(n)$ o número de operações executadas:

$$T(n) = \begin{cases} \Theta(1), & n = 1 \\ T(n-1) + \Theta(1), & n > 1. \end{cases}$$

A solução da recorrência é:

$$\sum_{1}^{n} \Theta(1) = n\Theta(1) = \Theta(n).$$



Subsequência consecutiva máxima (SCM)

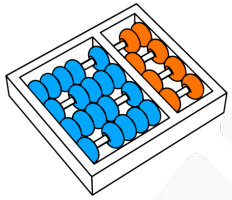
Problema

Entrada: *Uma sequência* $X = x_1, x_2, \ldots, x_n$ *de números reais.*

Saída: *Uma* ***subsequência consecutiva*** $Y = x_i, x_{i+1}, \ldots, x_j$, *onde* $1 \leq i, j \leq n$, *cuja soma seja máxima.*

Exemplos:

$X = [4, 2, -7, 3, 0, -2, 1, 5, -2]$	Resp: $Y = [3, 0, -2, 1, 5]$
$X = [-1, -2, 0]$	Resp: $Y = [0]$ ou $Y = [\,]$
$X = [-3, -1]$	Resp: $Y = [\,]$

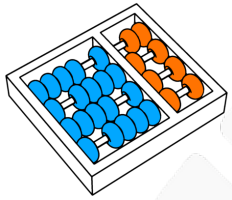


Hipótese

Teorema (Hipótese de indução)

Sabemos calcular a SCM *de sequências de comprimento* $n-1$.

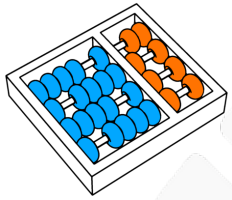
- Considere uma sequência $X = x_1, x_2, \ldots, x_n$.
- Removendo x_n, obtemos uma sequência menor X'.
- Usando a h.i., obtemos uma SCM $Y' = x_i, x_{i+1}, \ldots, x_j$ de X'.



Passo indutivo

Há três casos a analisar

1. $Y' = \emptyset$**:** se $x_n \geq 0$, faça $Y = x_n$, senão faça $Y = \emptyset$.
2. $j = n - 1$**:** se $x_n \geq 0$, faça $Y = Y'||x_n$, senão faça $Y = Y'$.
3. $j < n - 1$**:** consideramos dois sub-casos:
   - Se Y' for uma SCM de X, então basta fazer $Y = Y'$.
   - Caso contrário, x_n faz parte de uma SCM de X e fazemos $Y = x_k, x_{k+1}, \ldots, x_n$, para algum $k \leq n - 1$.

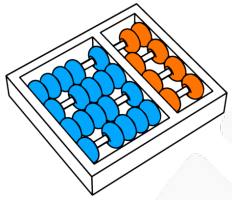


Refletindo

- Podemos executar os dois primeiros casos rapidamente.
- Para o último caso, a h.i. não fornece o valor de k.
- Nesse caso, sabemos que uma SCM é

$$Y = x_k, x_{k+1}, \ldots, x_{n-1}, x_n,$$

- ou seja, Y é um sufixo de X.
- Basta conhecer também o **sufixo de maior soma** de X'.

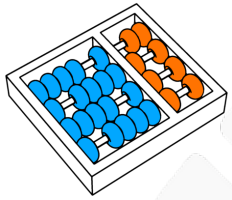


Hipótese de indução reforçada

Teorema (Hipótese de indução reforçada)

Sabemos calcular a SCM *e o sufixo de maior soma de sequências de comprimento* $n-1$.

- Consideramos também sequência de tamanho $n = 0$.
- Nesse caso, o sufixo de maior soma e a SCM são vazias.



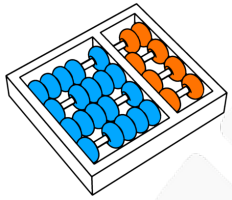
Entrada e saída do subproblema

Problema (Subproblema)

Entrada: *Um inteiro n e n números reais $X = x_1, x_2, \ldots, x_n$.*

Saída:

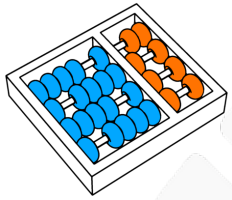
- *Inteiros i, j, k tais que:*
  - *$Y = x_i, x_{i+1}, \ldots, x_j$ é uma SCM de X.*
  - *$S = x_k, x_{k+1}, \ldots, x_n$ é um sufixo de soma máxima de X.*
- *Números reais $MaxSeq, MaxSuf$ tais que:*
  - *A soma de Y é $MaxSeq$.*
  - *A soma de S é $MaxSuf$.*



Algoritmo

Algoritmo: $\mathrm{SCM}(X, n)$

```
1  se n = 0
2      i, j, k ← 1  MaxSeq, MaxSuf ← 0
3  senão
4      i, j, k, MaxSeq, MaxSuf ← SCM(X, n − 1)
5      MaxSuf ← MaxSuf + x_n
6      se MaxSuf < 0
7          k ← n + 1,  MaxSuf ← 0
8      se MaxSuf > MaxSeq
9          i ← k,  j ← n,  MaxSeq ← MaxSuf
10 devolva i, j, k, MaxSeq, MaxSuf
```



Complexidade

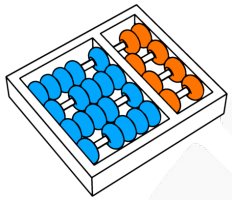
O tempo de execução $T(n)$ de SCM é:

$$T(n) = \begin{cases} \Theta(1), & n = 1 \\ T(n-1) + \Theta(1), & n > 1. \end{cases}$$

A solução desta recorrência é:

$$\sum_{1}^{n} \Theta(1) = n\Theta(1) = \Theta(n).$$

Divisão e conquista



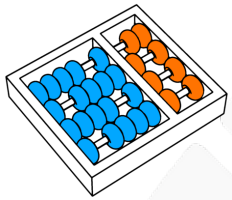
Projeto de algoritmos por divisão e conquista

Relembre que um algoritmo de **divisão e conquista**:

- Divide uma instância do problema considerado em partes.
- Combina as várias soluções parciais.

Algumas características:

- Normalmente criamos pelo menos duas subinstâncias.
- Elas podem ser muito menores que a instância original.

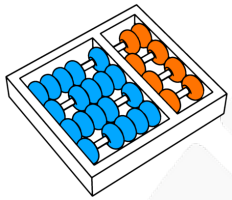


Algoritmo genérico

Algoritmo: DIVISAO-CONQUISTA(x)

```
1 se a instância x é suficientemente pequena
2     devolva SOLUCAO(x)
3 senão
4     decomponha x em instâncias menores x_1, x_2, ··· , x_k
5     para i de 1 até k
6         y_i ← DIVISAO-CONQUISTA(x_i)
7     combine as soluções y_i para obter uma solução y
8     devolva y
```

Alguns exemplos



Exponenciação

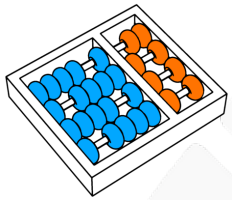
Problema

Entrada: *Um número real a e um inteiro $n \geq 0$.*

Saída: *Potência a^n.*

Um estratégia recursiva simples:

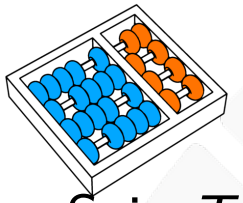
- Se $n = 0$:
  - Devolva $a^0 = 1$.
- Se $n > 0$:
  - Suponha que sabemos calcular a^{n-1}.
  - Calculamos $a^n = a^{n-1} \cdot a$.



Algoritmo com recursão simples

Algoritmo: EXPONENCIACAO(a, n)

```
1 se n = 0
2     an ← 1
3 senão
4     an' ← EXPONENCIACAO(a, n − 1)
5     an ← an' ∗ a
6 devolva an
```



Complexidade da recursão simples

Seja $T(n)$ o número de operações executadas:

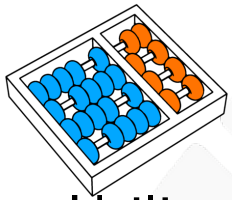
$$T(n) = \begin{cases} \Theta(1), & n = 0 \\ T(n-1) + \Theta(1), & n > 0, \end{cases}$$

A solução da recorrência é:

$$T(n) = \Theta(1) + \sum_{i=1}^{n} \Theta(1) = \Theta(n).$$

Pergunta: o algoritmo é linear no tamanho da entrada?

- O número de bits para representar n é $m = \lfloor \log_2 n \rfloor + 1$.
- Então o tempo do algoritmo é $\Theta(2^m)$.
- Que é **EXPONENCIAL** no tamanho da entrada.



Usando divisão e conquista

Utilizamos a mesma hipótese

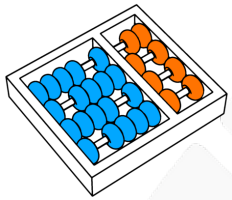
Teorema (Hipótese de indução)

Suponha que, para qualquer inteiro $n > 0$ e real a, sei calcular a^k, para todo $k < n$.

Mas agora dividimos a potência em várias partes:

- Se $n = 0$:
  - Devolva $a^0 = 1$.
- Se $n > 0$:
  - Calculamos recursivamente $a^{\lfloor \frac{n}{2} \rfloor}$.
  - Podemos calcular a potência da seguinte forma:

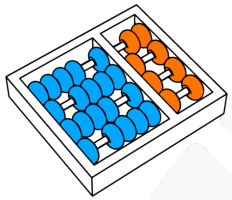
$$a^n = \begin{cases} \left(a^{\lfloor \frac{n}{2} \rfloor}\right)^2, & \text{se } n \text{ é par} \\ a \cdot \left(a^{\lfloor \frac{n}{2} \rfloor}\right)^2, & \text{se } n \text{ é ímpar} \end{cases}$$



Exponenciação com divisão e conquista

Algoritmo: EXPONENCIACAO-DC(a, n)

```
1 se n = 0
2     an ← 1
3 senão
4     an′ ← EXPONENCIACAO-DC(a, ⌊n/2⌋)
5     an ← an′ · an′
6     se n é ímpar
7         an ← an · a
8 devolva an
```



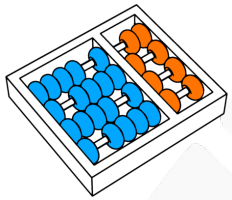
Complexidade

O tempo de execução é dado por:

$$T(n) = \begin{cases} c_1, & n = 0 \\ T(\lfloor \frac{n}{2} \rfloor) + c_2, & n > 0, \end{cases}$$

A solução da recorrência é $T(n) = \Theta(\log n)$.

- Esse algoritmo é linear no número de bits da entrada!



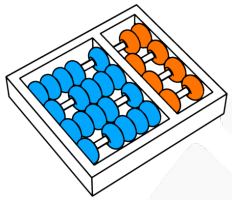
Busca binária

Problema

Entrada: *Um vetor de números ordenado A com n elementos e um número x.*

Saída: *Algum índice i tal que $A[i] = x$ se x estiver no vetor, ou* NIL *se x não estiver no vetor.*

- Se utilizássemos uma recursão simples, teríamos um algoritmo linear. (qual seria a hipótese de indução?)
- Vamos decompor o vetor em duas partes com cerca de **metade** dos elementos.



Entrada e saída

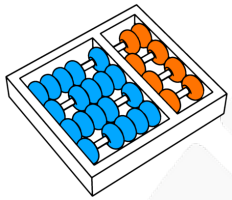
Como representar subvetores?

Problema (Problema generalizado)

Entrada:

- *Vetor A com n números.*
- *Índice e do elemento mais à esquerda do subvetor.*
- *Índice d do elemento mais à direita do subvetor.*
- *Número x.*

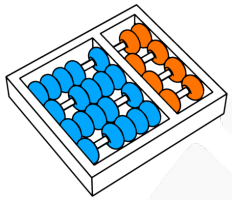
Saída: *Algum índice i tal que $A[i] = x$ se x estiver no vetor, ou* NIL *se x não estiver no vetor.*



Algoritmo

Algoritmo: BUSCA-BINARIA(A, e, d, x)

```
1 se e > d
2     i ← NIL
3 senão
4     i ← ⌊(e+d)/2⌋
5     se A[i] > x
6         i ← BUSCA-BINARIA(A, e, i − 1, x)
7     se A[i] < x
8         i ← BUSCA-BINARIA(A, i + 1, d, x)
9 devolva i
```



Complexidade

O tempo de execução é:

$$T(n) = \begin{cases} c_1, & n = 1 \\ T(\lceil \frac{n}{2} \rceil) + c_2, & n > 1, \end{cases}$$

Resolvendo a recorrência, temos:

$$T(n) = \Theta(\log n)$$

- Esse algoritmo é melhor que a busca linear.
- Se o vetor **NÃO** estivesse ordenado, qual seria melhor?

Projeto de algoritmos recursivos. Divisão e conquista

MO417 - Complexidade de Algoritmos I

04/24

8

Santiago Valdés Ravelo
https://ic.unicamp.br/~santiago/
ravelo@unicamp.br

UNICAMP