Resumo bomba de Compleza

Algoritmos e Complexidade

Algoritmo: Procedimento descrito passo-a-passo para resolução de um problema.

Técnicas para Algoritmos Interativos

1° Indentificar pré-condições, identificar pós-cond.
2° Indentificar os laços!
   - ↳ Condições de permanência no laço.
   - ↳ Condições são eventualmente atendidas (Finitude!)
   - ↳ Lei de recorrência (ou regra de transição).
3° Encontrar um Invariante!
   - ↳ Mostrar que no final o invariante é correto.

Complexidade

É verificar a quantidade de vezes que o programa 'roda' relacionada à entrada. Isso se denomina 'custo'.

em: $\sum_{i=0}^{n-1}(i+1) = \frac{n(n+1)}{2} = \frac{n^2+n}{2}$, logo o custo deverá ser quadrático em relação a x.

Já se n tem um custo fixo para cada x diz-se que é proporcional à x.

Ordenação

- Por seleção (em ordem decrescente):
  - ↳ $\frac{n^2-n}{2}$. (quadrática no tamanho de n).
- Por Inserção:
  - ↳ Custo varia com a entrada.
    - ↳ Pior caso $\frac{n^2-n}{2}$
    - ↳ melhor caso: $\sim n$.

Análise Assintótica → (valores grandes de N)

↳ Quantifica a Complexidade usando o crescimento de Tempo em função da entrada.

- Assumindo tamanho N da entrada:
- Notação "BigOh"
  - ↳ $\mathcal{O}(N)$: ordem linear
  - ↳ $\mathcal{O}(N^2)$: ordem quadrática
  - ↳ $\mathcal{O}(2^N)$: ordem ~~logarítmica~~ exponencial
  - ↳ $\mathcal{O}(\log N)$: Ordem logarítmica.

⊛ Definição formal slide 61 - aula 2017.

propriedades:
- $\mathcal{O}(N^3) + \mathcal{O}(N^2) = \mathcal{O}(N^3)$ → maior complex. domina.
- $\mathcal{O}(N^2) + \mathcal{O}(N) = \mathcal{O}(N^2)$
- $\mathcal{O}(\sum_{i=0}^{k} a_i N^i) = \mathcal{O}(N^k)$
- $\mathcal{O}(1)$ trecho de complexidade constante.

* A complexidade é considerada no pior caso.

exemplos:

```
for i in range(1,n):
    sum += 1
```
custo: $\mathcal{O}(N)$

```
for i in range(1,n):
    for j in range(1,n):
        sum += 1
```
custo: $\mathcal{O}(N^2)$.

Resumindo: * Avaliar e analisar pontos críticos focando em laços.
* Otimizar pontos críticos somente após verificar se o algoritmo funciona.

- busca sequencial → $\mathcal{O}(N)$
- busca binária ~ $\frac{N}{2}, \frac{N}{4}$ → $\mathcal{O}([\log N])$.

Recursão

↳ Um procedimento recursivo é um procedimento que chama a si mesmo durante a execução.
↳ Análise assintótica cuidadosa e perigosa.

exemplo: fatorial:

```
def fatorial(x): return 1 if x<=1 else
                        x * fatorial.
```

- O método funciona para x=1 e para x>1, se funciona para (x-1). Por indução finita calcula os fatoriais corretamente.

• O caso $x=1$, em que o algoritmo para é o caso base da recursão.

↳ quando a função não consegue executar ela mesma.

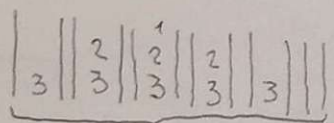
Controle via Stack

↳ procedimento recursivo que cria várias cópias de suas variáveis no Stack (pilha) a medida que suas chamadas são feitas.

* Recursivos podem "esconder" Laços.

ex:

Fatorial(3) → Fatorial(2) → fatorial(1)



stack

Recursão de Cauda (Tail call recursion)

↳ Quando a última operação de um corpo de uma função é a chamada de outra função, o estado não precisa ser preservado.

exemplo:

```
Function F(x):
  if C(x) then
    return E(x)
  else
    return F(G(x))
  end if
end function
```

é equivalente:

```
Function F(x):
  while Not C(x) do
    x ← G(x)
  end while
  returno E(x)
end function
```

• Não há custo adicional de armazenamento.

Divisão e Conquista

↳ Basicamente divide o problema até o caso base ser alcançado e ser trivialmente Resolvido.

Para uma relação de recorrência (qtd de chamadas recursivas):

$$T(N) = A \cdot T\left(\frac{N}{B}\right) + c \cdot f(N)$$

- A: quanto a entrada é dividida
- $f(N)$: custo de combinar subproblemas } complexidade não recursiva

• Um resultado geral:

$$T(N) = AT\left(\frac{N}{B}\right) + c \cdot N^L$$

$$T(N) = \begin{cases} O(N^{\log_B A}) & \text{se } A > B^L \\ O(N^L \log N) & \text{se } A = B^L \\ O(N^L) & \text{se } A < B^L \end{cases}$$