COMPLEXIDADE
PARTE II

Ana Neri (ana.i.neri@inesctec.pt)

DI,
Universidade do Minho

February 27, 2023

Conteúdo

1 Relembrar a aula anterior **2**

2 Definições recursivas **3**

2.1 Relação de recorrência 5

2.2 Árvore de recursão 7

3 Análise amortizada **9**

3.1 Análise Agregada 11

RELEMBRAR A AULA ANTERIOR

- Que fatores temos de ter em conta quando fazemos a analise temporal a um programa?
- Para o programa:

```
def sort_sel(lista):
    for i in range(len(lista)):
        min_idx = i
        for j in range(i+1, len(lista)):
            if lista[j] < lista[min_idx]:
                min_idx = j
        if i != min_idx:
            lista[i], lista[min_idx] = lista[min_idx], lista[i]
    return lista
```

  - Qual é o tamanho do input?
  - Quais as componentes atómicas?
  - Porque é que precisamos de componentes atómicas?
  - Qual será o melhor caso, o pior caso e o caso médio?
- Qual é a complexidade em notação $\mathcal{O}$-grande deste algoritmo.

Definições recursivas

Nos casos que vimos até agora o número de iterações no ciclo era controlado por uma variável.
Tal não acontece em todos os casos!
Para definir uma função de custo de forma recursiva vamos usar **equações de recorrência**.
Vamos ver isto passo a passo com um exemplo.

Definições recursivas

```
def maxInd (v, N): #v é um vetor com comprimento N
    int i
    if (N==1):
        i=0
    else:
        i=maxInd(v, N-1)
        if (v[N-1]>v[i]):
            i =N-1
    return i
```

De é que depende o custo?
O custo depende pode ser definido como o número de comparações entre elementos do array.

Definição recursiva de $T(N)$
Relação de recorrência

Relação de recorrência:

$$T(N) = \begin{cases} 0 & \text{se} N = 1 \\ \underbrace{T(N-1)}_{\texttt{maxInd(v,N-1)}} + 1 & \text{se} N > 1 \end{cases} \quad (1)$$

Esta definição pode ser *resolvida*:

$$\begin{aligned} T(1) &= 0 \\ T(2) &= 1 + T(1) = 1 + 0 = 1 \\ T(3) &= 1 + T(2) = 1 + 1 + 0 = 2 \\ T(4) &= 1 + T(3) = 1 + 1 + 1 + 0 = 3 \\ \ldots \\ T(N) &= \underbrace{1 + 1 + \cdots + 1 + 0}_{N-1 \text{text}} \\ &= N - 1 \end{aligned} \quad (2)$$

Relação recursiva

Exemplo

```
def bsearch(arr, s, N, x):
  i=-1
  if N >= s:
    mid = (N + s) // 2
    if arr[mid] == x:
      i = mid
    elif arr[mid] > x:
      i = bsearch(arr, s, mid - 1, x)
    else:
      i = bsearch(arr, mid + 1, N, x)
  return i
```

Calculemos então $T(N)$ como o número de comparações feitas por esta função quando se procura um elemento que não existe (pior caso) num array com N elementos.

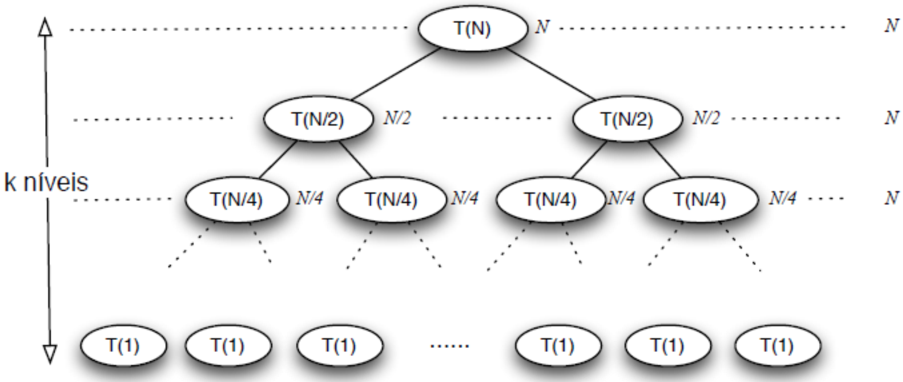
Definição Recursiva

Árvore de recursão

Quando a analise de programas não é linear podemos induzir resultados *desenhando* a expansão dos termos sob a forma de uma árvore.

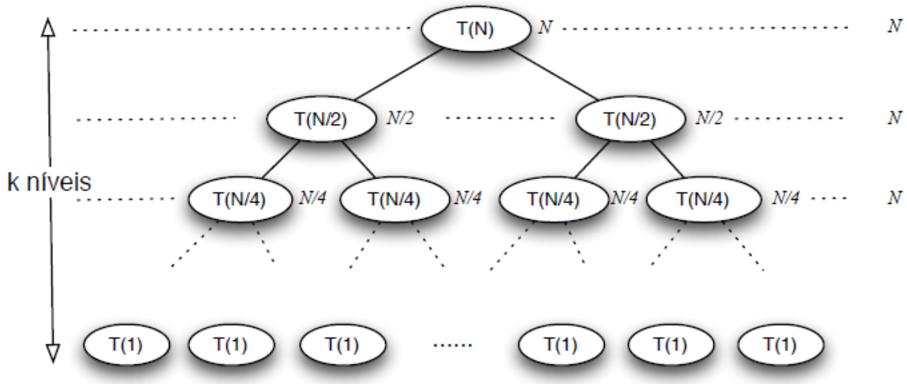
$$T(N) = \begin{cases} c_1 & \text{se} N = 0 \\ N + 2 * T\left(\frac{N}{2}\right) & \text{se} N > 0 \end{cases} \tag{3}$$

Para um N genérico e suficientemente grande temos:



Definição Recursiva

Árvore de recursão



$$\begin{aligned} T(N) &= \log_2(N) * N + 2^{1+\log_2(N)} * c_1 \\ &= \log_2(N) * N + 2 * N * c_1 \end{aligned} \tag{4}$$

Análise amortizada

```
def inc(b):
  N=len(b)
  i=N-1
  while((i>=0) and (b[i] == 1)):
    b[i] = 0
    i-=1
  if (i>=0) :
    b[i] = 1
  return b
```

A complexidade:

1. no melhor caso é 1.
2. no pior caso é $N-1$.
3. no caso médio é
   $\overline{T}(N) = \left(\sum_{k=1}^{N} \frac{k}{2^k}\right) + \frac{N}{2^N} = 2 - \frac{1}{2^{N-1}}$

Como $\lim \overline{T}(N) = 2$ temos $\overline{T}(N) = \Theta(1)$
Nesta caso, há uma grande diferença entre o caso médio e o pior caso porque o pior caso tem pouca probabilidade de acontecer.

Análise amortizada

Este tipo de situações leva-nos à **análise amortizada**, onde invés de analisar o custo de uma operação vamos analisar o custo de uma sequência de operações.

Tipicamente analisamos a sequência com o pior custo.

Com N operações $\{op_i\}_{1 \leq i \leq N}$ cada uma com custo c_i.
Queremos o custo amortizado da operação i, denotado por $\hat{c}_i$, de forma a que a soma dos custos amortizados seja um limite superior da soma dos custos reais:

$$\sum_{i=1}^{N} \hat{c}_i \geq \sum_{i=1}^{N} c_i \tag{5}$$

Análise Agregada

Primeiro passo na análise amortizada é calcular a soma dos custos reais das operações da sequência.
Com isso calculamos o custo amortizado da sequência:

$$\hat{c}_i = \frac{1}{N} \sum_{i=1}^{N} c_i \tag{6}$$

Próximo episódio

- Análise amortizada
- Classes