# TP 02

Ana Neri

20 Fevereiro 2023

Exercícios práticos sobre complexidade. (A maior parte das solução veio do chat GTP pode não estar completamente certa.)

**Exercício 1** Considere a função que verifica se um vetor de inteiros contem elementos repetidos.

```
def repetidos(v, N):
    rep = 0
    i = 0
    while i < N-1 and not rep:
        j = i+1
        while j < N and v[i] == v[j]:
            rep = 1
            j += 1
        i += 1
    return rep
```

- Calcule o melhor caso;
- Calcule o pior caso;
- Para o pior caso definido acima, calcule o número de comparações (entre elementos do vetor) que são efetuadas (em função do tamanho do array argumento).

**Solução 1** O melhor caso ocorre quando todos os elementos do vetor são distintos, e a função apenas faz uma comparação de elementos do vetor. Nesse caso, a função vai percorrer todo o vetor sem encontrar elementos repetidos, e vai retornar 0.

O pior caso ocorre quando todos os elementos do vetor são iguais, e a função tem que comparar cada par de elementos do vetor. Nesse caso, a função vai percorrer a matriz comparando os elementos até encontrar um par repetido ou chegar ao final do vetor. Portanto, o número de comparações será dado pela soma dos primeiros $N-1$ números inteiros, ou seja, $(N-1)*N/2$.

Para o pior caso, como visto acima, a função vai fazer $(N-1)*N/2$ comparações de elementos do vetor.

**Exercício 2** Considera a função que calcula o número de elementos diferentes num array de inteiros.

```
def diferentes(v, N):
    dif = 0
    for i in range(N):
        j=i+1
        while j<N and v[i]!= v[j]:
            if j == N:
                dif += 1
    return dif
```

- Calcule o melhor caso;
- Calcule o pior caso;
- Para o pior caso definido acima, calcule o número de comparações (entre elementos do vetor) que são efetuadas (em função do tamanho do array argumento).

**Solução 2** Melhor caso: O melhor caso ocorre quando todos os elementos do vetor são iguais, pois no primeiro teste realizado pelo loop interno, a condição de comparação $v[i] != v[j]$ é falsa e o loop é interrompido, evitando a realização de outras comparações desnecessárias. Nesse caso, o número de comparações é dado por 0, já que não é necessário fazer nenhuma comparação.

Pior caso: O pior caso ocorre quando todos os elementos do vetor são diferentes, pois nesse caso, todas as comparações terão que ser realizadas. Nesse caso, o número de comparações é dado por uma soma aritmética: $(N-1) + (N-2) + ... + 2 + 1$, que pode ser resumida como $(N^2 - N)/2$.

Para o pior caso definido acima, o número de comparações é dado por $(N^2 - N)/2$. Por exemplo, se $N = 4$, o número de comparações será $(4^2 - 4)/2 = 6$.

**Exercício 3** Na aula teórica vimos uma função que calculava o produto de dois números.

Considere agora um algoritmo alternativo para este cálculo.

```
def bprod(x, y):
    r = 0
    while x > 0:
        if x and 1:
            r = r + y
        x = x >> 1
        y = y << 1
    return r
```

Qual é o custo da função no pior caso?

**Solução 3**

O algoritmo alternativo apresentado é uma implementação do algoritmo de multiplicação de dois números em binário utilizando a técnica de shift-and-add.

Vamos considerar apenas o número de adições que são feitas à variável `r`. O tamanho N do input corresponde ao número de bits necessários para representar o inteiro x.

No pior caso todos os bits estão a 1, ou seja, o valor de $x_0$ é $2^N - 1$.

A cada iteração;

- 1 dos bits x passa a 0,
- é efetuada uma adição

Logo temos:

$$T(N) = N$$

**Exercício 4** Considere a seguinte definição de uma função que calcula a potência inteira de um número.

```
def pot(base, exp):
    r = 1.0
    while exp > 0:
        r *= base
        exp -= 1
    return r
```

Apresente uma versão alternativa desta função cujo número de multiplicações, no pior caso, seja proporcional ao número de bits usados para representar o expoente (Sugestão: use como inspiração as funções apresentadas na aula teórica e no exemplo anterior para calcular o produto de dois números).

**Solução 4**

```
    def pot(base, exp):
    r = 1.0
    while exp > 0:
        if exp and 1:
            r *= base
        exp = exp >> 1
        base *= base
    return r
```

**Exercício 5** Considere a seguinte definição da função `inc` que recebe um array de `N` bits (representando um inteiro `x`) e que modifica o array de forma a representar `x+1`.

```
def inc(x, N):
    i = N - 1
    while i >= 0 and x[i] == 1:
```

```
    i -= 1
    x[i] = 0
if i < 0:
    return 1
x[i] = 1
return 0
```

Calcule o custo médio desta função (em termos do numero de bit alterados).

**Solução 5**

A complexidade desta função pode ser reduzida ao número de bits alterados, que varia entre

- 1 no (melhor) caso de o bit menos significativo ser 0
- N no (pior) caso de os últimos $N-1$ bits serem 1.

Para determinarmos o número médio de bits que são alterados vamos assumir que o número representado no vetor de bits é um número aleatório. Isto é equivalente a dizer que cada posição do vetor pode ter, com a mesma probabilidade, os valores 0 ou 1.

Desta forma,

- a probabilidade de mudar apenas 1 bit é $\frac{1}{2}$ (corresponde à probabilidade de o bit menos significativo ser 0).
- a probabilidade de mudar apenas 2 bits é $\frac{1}{4}$ (corresponde à probabilidade de o bit menos significativo ser 1 e o seguinte ser 0).
- a probabilidade de mudar apenas 3 bits é $\frac{1}{8}$ (corresponde à probabilidade de os 2 bits menos significativos serem 1 e o seguinte ser 0).
- a probabilidade de mudar apenas k bits ($0 \leq k < N$) é $\frac{1}{2^k}$ (corresponde à probabilidade de os $k-1$ bits menos significativos serem 1 e o seguinte ser 0).
- Existem dois casos em que o número de bits mudados é N que correspondem às duas situações em que todos os $N-1$ bits menos significativos são 1. Nestes casos fazem-se sempre N alterações.

O número médio de bits alterados é por isso dado pela expressão

$$\overline{T}(N) = (\sum_{k=1}^{N} \frac{k}{2^k}) + \frac{N}{2^N} = 2 - \frac{1}{2^N - 1}$$

A razão de ser desta proximidade do caso médio e do melhor caso, ou por oposição, da diferença tão acentuada entre o caso médio e o pior caso, deve-se à pouca probabilidade de o pior caso acontecer.

**Exercício 6** Dado um vetor $v$ de $N$ números inteiros, a mediana do vetor define-se como o elemento do vetor em que

- existem no máximo $N/2$ elementos (estritamente) menores do que ele, e
- existem no máximo $N/2$ elementos (estritamente) maiores do que ele.

Se o vetor estiver ordenado, a mediana corresponde ao valor que está na posição $N/2$[1].

1. Considere a seguinte definição de uma função que calcula a mediana de um vetor com comprimento $N$ ímpar.

```
def mediana(v, N):
    i = -1
    m = N
    M = N
    while i<N and (m > N/2 or M > N/2):
        i = i + 1
        m, M = quantos(v, N, v[i])
    return v[i]
```

   Assumindo que a função `quantos` executa em tempo linear no comprimento do vetor de input, identifique o melhor e pior caso de execução da função mediana. Para cada um desses casos determine a complexidade assimptótica da função mediana.

2. Calcule a complexidade média da função apresentada na alínea anterior. Para isso, assuma que os valores do vetor são perfeitamente aleatórios e, por isso, que a probabilidade de o elemento numa qualquer posição do vetor ser a mediana é uniforme ($\frac{1}{N}$ ).

**Solução 6** 1. O melhor caso de execução da função mediana é quando o primeiro elemento do vetor é a mediana. Nesse caso, a função quantos será chamada apenas uma vez, com a mediana como parâmetro, e o ciclo será executado apenas uma vez, levando à saída do primeiro elemento do vetor. A complexidade assimptótica para esse caso é $O(1)$.

   O pior caso de execução da função mediana é quando a mediana é o último elemento do vetor. Nesse caso, a função quantos será chamada $N$ vezes, e o loop será executado $N$ vezes antes de encontrar a mediana. A complexidade assimptótica para esse caso é $O(N^2)$, considerando que a função quantos executa em tempo linear.

2. Para calcular a complexidade média da função mediana, precisamos calcular a probabilidade de cada elemento ser a mediana. Como assumimos que os valores do vetor são perfeitamente aleatórios, cada elemento tem a mesma probabilidade de ser a mediana, ou seja, $\frac{1}{N}$.

[1] Quando número $N$ é par devemos fazer uma média aritmética entre os 2 números centrais.

Podemos calcular a complexidade média da função mediana usando a fórmula:

$$\sum_{i=1}^{N} \frac{i}{N} \mathcal{O}(i) = \mathcal{O}\left(\sum_{i=1}^{N} \frac{i^2}{N}\right) = \mathcal{O}\left(\frac{N^2}{3}\right) = \mathcal{O}(N^2)$$

Portanto, a complexidade média da função mediana é $O(N^2)$. Note que estamos usando o resultado de que a soma dos quadrados dos $N$ primeiros números inteiros é $\frac{N(N+1)(2N+1)}{6}$, o que leva à simplificação $\sum_{i=1}^{N} \frac{i^2}{N} = \frac{N+1}{3}$.