

# MAC0121 – Algoritmos e Estruturas de Dados I

BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

SEGUNDO SEMESTRE DE 2023

Prova 1 – 16/11/2023

Nome completo: \_\_\_\_\_

NUSP: \_\_\_\_\_

Assinatura: \_\_\_\_\_

## Instruções:

1. Não destaque as folhas deste caderno.
2. Preencha o cabeçalho acima.
3. A prova pode ser feita a lápis. Cuidado com a legibilidade.
4. A prova consta de **3 questões**. Verifique antes de começar a prova se o seu caderno de questões está completo.
5. Não é permitido o uso de folhas avulsas para rascunho.
6. Não é permitido o uso de artefatos eletrônicos.
7. Não é permitida a consulta a livros, apontamentos ou colegas.
8. Não é necessário apagar rascunhos no caderno de questões.

**DURAÇÃO DA PROVA: 2 horas**

Questão	Nota
1	
2	
3	
Total	





**Q2 (3.0 pontos)** Diga qual será a saída do programa abaixo quando executado com os dígitos de seu número USP, dados um a um, separado por espaços. Por exemplo, se seu NUSP fosse 31415926, você teria de simular o programa com entrada

3 1 4 1 5 9 2 6

**Importante:** seu rascunho deve dar alguma indicação de como você chegou a sua resposta.

```
public class SubsequenceMod2
{
    public static long noOccurrences(int[] s, int[] t) {
        return noOccurrences(s, 0, t, 0);
    }

    public static long noOccurrences(int[] s, int i, int[] t, int j) {
        int M = s.length;
        int N = t.length;
        if (i == M) return 1;
        if (j == N) return 0;

        if (s[i] == t[j])
            return noOccurrences(s, i + 1, t, j + 1)
                + noOccurrences(s, i, t, j + 1);
        return noOccurrences(s, i, t, j + 1);
    }

    public static void show(int[] v) {
        for (int i = 0; i < v.length; i++)
            StdOut.print(v[i]);
        StdOut.println();
    }

    public static void main(String[] args)
    {
        int[] NUSP = StdIn.readAllInts();

        for (int i = 0; i < NUSP.length; i++)
            NUSP[i] = NUSP[i] % 2;
        show(NUSP);

        int[] s1 = new int[1];
        s1[0] = NUSP[0];
        StdOut.println(noOccurrences(s1, NUSP));

        int[] s3 = new int[3];
        for (int i = 0; i < 3; i++)
            s3[i] = NUSP[i];
        show(s3);
        StdOut.println(noOccurrences(s3, NUSP));
    }
}
```

## Rascunho

## Saída do programa

**Q3 (4.0 pontos)** A função `noOccurrences(int[] s, int[] t)` da Questão 2 não é muito eficiente. Considere o seguinte programa incompleto:

```
public class SubsequencesQ3
{
    public static long noOccurrencesFast(int[] s, int[] t) {
        // a completar
    }

    public static void main(String[] args)
    {
        int[] t = StdIn.readAllInts();

        int M = Integer.parseInt(args[0]);
        int[] s = new int[M];
        for (int i = 0; i < M; i++)
            s[i] = t[i];

        Stopwatch sw = new Stopwatch();
        StdOut.println(noOccurrencesFast(s, t)
            + " [" + sw.elapsedTime()+ " seconds]");
        sw = new Stopwatch();
        StdOut.println(SubsequenceMod2.noOccurrences(s, t)
            + " [" + sw.elapsedTime()+ " seconds]");
    }
}
```

Note que o programa acima usa a função `noOccurrences()` da Questão 2 e ele contém uma função alternativa (incompleta) `noOccurrencesFast()`. Uma das coisas que você deve fazer nesta questão é implementar `noOccurrencesFast()`. Com a implementação esperada de `noOccurrencesFast()`, teríamos o seguinte tipo de comportamento de `SubsequencesQ3.java`:

```
$ java-introcs RandomInts 10000 200 121121 | java-introcs SubsequencesQ3 4
294819 [0.002 seconds]
294819 [0.477 seconds]
$ java-introcs RandomInts 10000 200 121121 | java-introcs SubsequencesQ3 5
2469867 [0.002 seconds]
2469867 [4.297 seconds]
$ java-introcs RandomInts 10000 200 121121 | java-introcs SubsequencesQ3 6
35180314 [0.002 seconds]
35180314 [34.202 seconds]
$
```

Acima, `RandomInts.java` gera 10000 inteiros no intervalo  $[0, 200)$ , usando 121121 como a semente do gerador de números aleatórios de `StdRandom.java` (os detalhes de implementação de `RandomInts.java` não são muito relevantes). Os exemplos de execução acima mostram que `noOccurrencesFast()` é muito mais rápido que `SubsequenceMod2.noOccurrences()`. Naturalmente, queremos também que `noOccurrencesFast(s, t)` devolva o mesmo valor que `SubsequenceMod2.noOccurrences(s, t)`.







(Rascunho)