CCM0118 – Computação I

Curso de Ciências Moleculares

Segundo Semestre de 2024

Prova 2 – 5/12/2024

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. **A prova vale 12 pontos**.
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	

Q1 (4.0 pontos) Considere o seguinte programa.

```
public class Q1 {

    public  static void subseq(String s, int k) {
        boolean[] inPrefix = new boolean[128];
        subseq("", s, k, inPrefix);
    }

    public static void subseq(String prefix, String s, int k, boolean[] inPrefix) {
        if (k == 0) {
            StdOut.println(prefix);
            return;
        }
        for (int i = 0; i < s.length(); i++) {
            char x = s.charAt(i);
            if (!inPrefix[x]) {
                inPrefix[x] = true;
                subseq(prefix + s.charAt(i), s.substring(i + 1), k - 1, inPrefix);
                inPrefix[x] = false;
            }
        }
    }

    public static void main(String[] args) {
        String sequence = args[0];
        int k = Integer.parseInt(args[1]);
        subseq(sequence, k);
    }
}
```

(a) Diga qual será a saída das seguintes execuções:

(*i*) Execução 1:

```
$ java-introcs Q1 abc 2
```

(*ii*) Execução 2:

```
$ java-introcs Q1 aba 2
```

(*iii*) Execução 3:

```
$ java-introcs Q1 aaa 2
```

(*iv*) Execução 4:

```
$ java-introcs Q1 aabb 2
```

(b) Quantos strings serão impressos no total com a entrada $s = a^n b^n c^n$ e 3? Aqui, n é um número natural positivo, e s é o string com n ocorrências de a, seguida de n ocorrências de b, seguida de n ocorrências de c. Naturalmente, sua resposta deve ser em função de n.

Observação. Para lembrar certas coisas de Java, você pode achar útil considerar o programa abaixo. Uma execução deste programa é dada mais à frente.

```
public class Example
{
    public static void main(String[] args)
    {
        String seq = args[0];
        boolean[] there = new boolean[128];

        char x = seq.charAt(0);
        StdOut.println(there[x]);
        there[x] = true;
        StdOut.println(there[x]);

        StdOut.println(seq);
        // sufixo começando com a letra seq.charAt(2)
        String seq2 = seq.substring(2);
        StdOut.println(seq2);
    }
}
```

Exemplo de execução:

```
$ java-introcs Example abcde
false
true
abcde
cde
$
```

Q2 (4.0 pontos) Considere a função `power()` abaixo.

```
public static int power(int a, int b) {
    if (b == 0)
        return 1;
    if (b % 2 == 0) {
        StdOut.print(".");
        return power(a * a, b / 2);
    } else {
        StdOut.print("..");
        return power(a * a, b / 2) * a;
    }
}
```

(a) Que valor devolve a chamada `power(3, 2)`?

(b) Que valor devolve a chamada `power(2, 5)`?

(c) Suponha que a e b sejam inteiros com $a > 0$ e $b \geq 0$. O que é o valor devolvido pela chamada `power(a, b)`? (Ignore problemas de *overflow*.)

(d) Justifique sua resposta para o item (c) acima, fazendo indução em b.

(e) Suponha que temos de pagar R$1 por cada multiplicação que é feita por `power()`. Quanto temos de pagar pela execução de `power(3, 32)`? Note que o número de `'.'` impressos por `power()` é quanto devemos pagar. Esboce como você chegou a sua resposta.

Q3 (4.0 pontos) Considere o seguinte programa incompleto:

```
public class Q3
{
    public static boolean occurs(String s, String t) {
        if (s.length() == 0)
            return true;
        if (t.length() == 0)
            return false;
        if (s.charAt(0) == t.charAt(0))
            return occurs(s.substring(1), t.substring(1));
        else
            return occurs(s, t.substring(1));
    }

    public static int noOccs(String s, String t) {
        if (s.length() == 0)
            return 1;
        if (t.length() == 0)
            return 0;
        if (s.charAt(0) == t.charAt(0))
            return noOccs(s.substring(1), t.substring(1)) +
                noOccs(s, t.substring(1));
        else
            return noOccs(s, t.substring(1));
    }

    public static int noOccsFast(String s, String t) {
        int M = s.length();
        int N = t.length();

        // occ[i][j] = number of occurrences of s[i, M) in t[j, N)
        int[][] occ = new int[M + 1][N + 1];

        // a completar

        return occ[0][0];
    }



    public static void main(String[] args) {
        String s = args[0];
        String t = args[1];
        StdOut.println("s = " + s);
        StdOut.println("t = " + t);
        StdOut.println("s occurs in t: " + occurs(s, t));
        Stopwatch sw = new Stopwatch();
        StdOut.println("Number of occurrences of s in t (fast): " + noOccsFast(s, t));
        StdOut.println("[elapsed time: " + sw.elapsedTime() + "]");
```

```
        sw = new Stopwatch();
        StdOut.println("Number of occurrences of s in t (slow): " + noOccs(s, t));
        StdOut.println("[elapsed time: " + sw.elapsedTime() + "]");
    }
}
```

Considere as seguintes execuções de `Q3.java`:

```
$ java-introcs Q3 cde aaeaddececbdccd
s = cde
t = aaeaddececbdccd
s occurs in t: false
Number of occurrences of s in t (fast): 0
[elapsed time: 0.001]
Number of occurrences of s in t (slow): 0
[elapsed time: 0.0]
$ java-introcs Q3 cde aaeaddececbdccdbdeeb
s = cde
t = aaeaddececbdccdbdeeb
s occurs in t: true
Number of occurrences of s in t (fast): 20
[elapsed time: 0.002]
Number of occurrences of s in t (slow): 20
[elapsed time: 0.0]
$
```

As execuções acima ilustram que `noOccsFast()` calcula o mesmo valor que `noOccs()`. Entretanto, os exemplos abaixo mostram que `noOccsFast()` é muito mais rápido que `noOccs()`.

```
$ java-introcs Q3 aaaaaaaaaaaaaa aaaaaaaaaaaaaaaaaaaaaaaaaaaa
s = aaaaaaaaaaaaaa
t = aaaaaaaaaaaaaaaaaaaaaaaaaaaa
s occurs in t: true
Number of occurrences of s in t (fast): 155117520
[elapsed time: 0.001]
Number of occurrences of s in t (slow): 155117520
[elapsed time: 13.755]
$ java-introcs Q3 aaaaaaaaaaaaaaa aaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
s = aaaaaaaaaaaaaaa
t = aaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
s occurs in t: true
Number of occurrences of s in t (fast): 601080390
[elapsed time: 0.001]
Number of occurrences of s in t (slow): 601080390
[elapsed time: 54.224]
$
```

(a) Diga quanto é o valor de `noOccs("aba", "ababa")`. Esboce como você chegou a sua resposta.

(b) Suponha que $s = s_1 \dots s_M$ e $t = t_1 \dots t_N$ sejam strings de comprimento M e N (isto é, s é um string com caracteres $s_1, s_2, \dots$ e analogamente para t). Diga precisamente o que é o valor de `noOccs(s, t)`.

(c) Complete a implementação de `noOccsFast()`: escreva o código que deve ser inserido no ponto indicado por "a completar". Queremos que `noOccsFast()` devolva o mesmo valor que `noOccs()`, mas que ele seja extremamente mais rápido (como ilustram os exemplos acima).

* * *

(Rascunho)

(Rascunho)