MAC0121 – Algoritmos e Estruturas de Dados I

Bacharelado em Ciência da Computação

Segundo Semestre de 2023

Prova 2 – 14/12/2023

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. **A prova vale 11 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) 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
```

Lembre que seu NUSP *não é* 31415926 `:-)`.

Importante: seu rascunho deve indicar claramente como você chegou a sua resposta.

```
public class Q1
{
    public static class Node
    {
        int item;
        Node next;
    }

    public static Node mystery(int x, Node f) {
        if (f == null) {
            f = new Node();
            f.item = x;
            f.next = null;
            return f;
        }
        f.next = mystery(x, f.next);
        return f;
    }

    public static void show(Node f) {
        for (Node t = f; t != null; t = t.next)
            StdOut.print(t.item + " ");
        StdOut.println();
    }

    public static Node mystery2(Node l) {
        if (l == null || l.next == null)
            return l;
        Node t = l, tt = l.next;
        l = mystery2(tt);
        tt.next = t;
        t.next = null;
        return l;
    }

    public static void main(String[] args)
    {
        Node first = null;
        while (!StdIn.isEmpty()) {
            int x = StdIn.readInt();
            first = mystery(x, first);
        }
        show(first);
        first = mystery2(first);
        show(first);
    }
}
```

Rascunho

Saída do programa

Q2 (4.0 pontos) Considere o seguinte programa:

```
public class Q2
{
    public static int occ(char x, String s) {
        int t = 0;
        for (int i = 0; i < s.length(); i++)
            if (s.charAt(i) == x)
                t++;
        return t;
    }

    public static void prDots(int N) {
        for (int i = 0; i < N; i++)
            StdOut.print(".");
        StdOut.println();
    }

    public static int ba(String s) {
        int N = s.length();
        // prDots(N);
        if (N <= 1)
            return 0;
        String l = s.substring(0, N/2);
        String r = s.substring(N/2, N);
        int B = occ('b', l);
        int A = occ('a', r);
        return B * A + ba(l) + ba(r);
    }

    public static int baPlain(String s) {
        int t = 0;
        for (int i = 0; i < s.length(); i++)
            for (int j = i + 1; j < s.length(); j++)
                if (s.charAt(i) == 'b' && s.charAt(j) == 'a')
                    t++;
        return t;
    }

    public static void main(String[] args)
    {
        String s = StdIn.readString();
        Stopwatch sw0 = new Stopwatch();
        StdOut.println(ba(s));
        StdOut.println("Elapsed time: " + sw0.elapsedTime());
        Stopwatch sw1 = new Stopwatch();
        StdOut.println(baPlain(s));
        StdOut.println("Elapsed time: " + sw1.elapsedTime());
    }
}
```

Suponha que temos também o programa `Generator.java` que gera palavras (*strings*) aleatórias sobre o alfabeto $\{a, b\}$, como no seguinte exemplo:

```
$ java-introcs Generator 10 8888
bbaaaaaaab
$
```

Na execução acima, o argumento 10 indica que uma palavra de comprimento 10 deve ser gerada e 8888 deve ser usado como a semente do gerador de números aleatórios.

Quando executamos `Q2.java` com entradas geradas por `Generator.java`, vemos o seguinte comportamento:

```
$ java-introcs Generator 10 8888 | java-introcs Q2
14
Elapsed time: 0.0
14
Elapsed time: 0.0
$ java-introcs Generator 50000 8888 | java-introcs Q2
315877158
Elapsed time: 0.014
315877158
Elapsed time: 2.428
$ java-introcs Generator 100000 8888 | java-introcs Q2
1255053679
Elapsed time: 0.011
1255053679
Elapsed time: 10.302
$
```

(a) As funções `ba()` e `baPlain()` determinam um certo valor associado à palavra fornecida `s`. Diga precisamente o que é esse valor.

(b) Justifique que `ba()` de fato determina o valor como você especificou no item anterior. (Deve ser claro que `baPlain()` determina tal valor.)

(c) Vemos nos exemplos de execução acima que a função `ba()` é bem mais eficiente que `baPlain()`. Para analisar a complexidade de tempo de `ba()`, imagine que você descomentou a linha de impressão `prDots(N)` em `ba()`. Vale que o tempo de execução da chamada `ba(s)` é proporcional ao número **total** de caracteres `'.'` impressos pela chamada `ba(s)`. Determine quantos caracteres `'.'` são impressos **no total** pela chamada `ba(s)`; dê sua resposta em função do comprimento N de `s`. Justifique sua resposta. **Observação.** Por simplicidade, neste item, suponha que a palavra de entrada `s` tenha comprimento $N = 2^n$ para um inteiro $n \geq 0$.

(d) Seja $T_{\texttt{ba}}(N)$ o tempo de execução da chamada `ba(s)` para `s` de comprimento N. Com base no item anterior, qual é a ordem de grandeza/crescimento de $T_{\texttt{ba}}(N)$ (não é necessário justificar sua resposta).

(e) Seja $T_{\texttt{baPlain}}(N)$ o tempo de execução da chamada `baPlain(s)` para `s` de comprimento N. Qual é a ordem de grandeza/crescimento de $T_{\texttt{baPlain}}(N)$? Justifique brevemente sua resposta.

Q3 (3.0 pontos) Escreva uma função de assinatura

```
public static boolean wellFormed(String expr)
```

que recebe uma expressão `expr` de parênteses e colchetes e decide se ela é uma expressão bem-formada. Veja alguns exemplos abaixo:

(i) Se `expr = ()([])`, então `wellFormed(expr)` deve valer `true`.

(ii) Se `expr = ][([])`, então `wellFormed(expr)` deve valer `false`.

(iii) Se `expr = ()([)]`, então `wellFormed(expr)` deve valer `false`.

(vi) Se `expr = ()(([])`, então `wellFormed(expr)` deve valer `false`.

Sua função deve necessariamente usar um objeto de tipo `Stack<Character>`. De fato, você deve completar o seguinte esqueleto:

```
public static boolean wellFormed(String expr) {
    Stack<Character> s = new Stack<>();
    int N = expr.length();

    // a completar

}
```

Observação. Formalmente, uma expressão de parênteses e colchetes é *bem-formada* se ela for da forma `(s)`, `[s]`, ou `st`, onde `s` e `t` são expressões bem-formadas. Ademais, a expressão vazia é também bem-formada.

(Rascunho)

Escreva aqui sua função `wellFormed()` (não deixe de reescrever sua assinatura etc).

* * *

(Rascunho)