CCM0128 – Computação II

Curso de Ciências Moleculares

Primeiro Semestre de 2025

Prova 2 – 26/6/2025

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 programa abaixo.

```
public class Q1 {

    // partition the subarray a using 3-way partitioning
    private static void partition3way(int[] a) {
        int N = a.length;
        int lt = 0, gt = N - 1;
        int v = a[0];
        int i = 1;
        show(a);
        // a[i .. gt] not examined yet
        // a[0 .. lt) < v, a[lt .. i) = v and v < a(gt .. N)
        while (i <= gt) {
            int cmp = a[i] - v;
            if (cmp < 0) {
                exch(a, lt++, i++);
                show(a);
            } else if (cmp > 0) {
                exch(a, i, gt--);
                show(a);
            } else
                i++;
        }
        // at the end: a[0 .. lt) < v = a[lt .. gt] < a(gt .. N).
    }

    private static void exch(int[] a, int i, int j) {
        int swap = a[i];
        a[i] = a[j];
        a[j] = swap;
    }

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

    public static void main(String[] args) {
        int[] a = StdIn.readAllInts();
        partition3way(a);
        StdOut.print("main: ");
        show(a);
    }
}
```

Se você executar o programa `Q1` acima com a entrada `4 1 5 9 2 6 4`, o começo da saída será

```
4 1 5 9 2 6 4
1 4 5 9 2 6 4
1 4 4 9 2 6 5
...
```

Para entender a evolução do processo de partição, você pode achar interessante adicionar '`|`' em seus rascunhos, como segue:

```
|4|1 5 9 2 6 4|
 1|4|5 9 2 6 4|
 1|4 4|9 2 6|5
...
```

Monte uma entrada s para o programa `Q1` da seguinte forma. Considere os 5 últimos dígitos de seu NUSP. Adicione no começo e no fim o dígito 4. Por exemplo, se seu NUSP fosse 31415926, a sequência s seria `4 1 5 9 2 6 4` (que é o exemplo acima).

(a) Monte s, usando seu NUSP: _____

(b) Diga qual será a saída do programa `Q1` quando executado com a entrada s de (a).

Rascunho

Rascunho

Saída do programa

Q2 (4.0 pontos) Considere o seguinte programa.

```
public class Q2 {

    public static void main(String[] args) {

        // Build array of strings, with each entry being a line of file
        // called args[0].  Thus, text[i] is the ith line of file args[0].
        In in = new In(args[0]);
        String[] text = in.readAllLines();

        ST<String, SET<Integer>> st = new ST<String, SET<Integer>>();

        int[] NUSP = StdIn.readAllInts();
        int L = NUSP.length;

        // Build new text, selecting lines from the original text
        // using NUSP digits.
        String newText = "";
        for (int i = 0; i < L; i++) {
            newText += text[NUSP[i]] + " ";
            StdOut.println(text[NUSP[i]]);
        }

        // Build array of the words in newText.  Thus, words[i] is
        // the ith word in newText.
        String[] words = newText.split("\\s+");

        // build word index of newText
        for (int i = 0; i < words.length; i++) {
            String s = words[i];
            if (!st.contains(s))
                st.put(s, new SET<Integer>());
            SET<Integer> set = st.get(s);
            set.add(i);
        }

        String s = "pedra";
        StdOut.print("pedras: ");
        StdOut.println(st.get(s));
    }
}
```

Para resolver esta questão, você pode achar conveniente saber que obtemos a saída

```
{ 0, 1, 2, 3, 4 }
```

ao executarmos o seguinte programa:

```
public class Q2Example
{
    public static void main(String[] args)
    {
        SET<Integer> s = new SET<>();
        for (int i = 4; i >= 0; i--)
            s.add(i);
        StdOut.println(s);
    }
}
```

Suponha agora que um arquivo chamado `pedra.txt` tem o seguinte conteúdo:

```
no meio do caminho tinha uma pedra
tinha uma pedra no meio do caminho
tinha uma pedra
no meio do caminho tinha uma pedra
nunca me esquecerei desse acontecimento
na vida de minhas retinas tão fatigadas
nunca me esquecerei que no meio do caminho
tinha uma pedra
tinha uma pedra no meio do caminho
no meio do caminho tinha uma pedra
```

Considere a execução de `Q2`

```
$ java-introcs Q2 pedra.txt
```

com os dígitos de seu NUSP como entrada. (Por exemplo, se seu NUSP fosse 31415926, a entrada seria `3 1 4 1 5 9 2 6`.) Diga qual será a saída da execução de `Q2` acima com os dígitos de seu NUSP como entrada. **Importante.** Seu NUSP não é 31415926.

Rascunho

Rascunho

Saída do programa

Q3 (4.0 pontos) Seja x uma sequência $x_0, x_1, \ldots, x_{N-1}$ de inteiros. Vamos dizer neste exercício que x é *crescente* se $x_0 \leq x_1 \leq \cdots \leq x_{N-1}$ e que x é *decrescente* se $x_0 > x_1 > \cdots > x_{N-1}$. Como usual, uma *subsequência* de x é uma sequência da forma $x_{i_0}, x_{i_1}, \ldots, x_{i_{k-1}}$, onde $k \geq 0$ e $0 \leq i_0 < i_1 < \cdots < i_{k-1} < N$.

(a) Considere a seguinte sequência x de inteiros:

```
-6 -1 -6 9 12 0 14 7 11 3 5 8 19 15 10 23 23 18 19 26 19 29 19 25 24 19 20 29 36 34 27
```

Note que uma subsequência crescente da sequência acima é

```
-6 -1 0 3 5 8 15 18 19 19 19 19 20 29 36
```

Seja y a subsequência acima. Prove que y é uma LIS de x (*longest increasing subsequence*; subsequência crescente mais longa). Isto é, prove que x não tem uma subsequência crescente com mais elementos que y.

(b) Considere o programa abaixo.

```
public class RandomSeq
{
    public static void main(String[] args)
    {
        int N = Integer.parseInt(args[0]);
        long seed = Long.parseLong(args[1]);
        StdRandom.setSeed(seed);
        for (int i = 0; i < N; i++)
            StdOut.print(StdRandom.uniformInt(1000) + " ");
        StdOut.println();
    }
}
```

Suponha que executamos `RandomSeq` com argumentos `N = 1024` e `seed = 128128`. Seja s a sequência gerada por esta execução de `RandomSeq`. Prove que vale pelo menos uma das três alternativas abaixo:

(i) s tem uma subsequência crescente com 33 elementos,

(ii) s tem uma subsequência decrescente com 33 elementos, ou

(iii) s tem **tanto** uma subsequência crescente de 32 elementos **como** uma subsequência decrescente de 32 elementos.

(Rascunho)

(Rascunho)