MAC0122 – Princípios de Desenvolvimento de Algoritmos

Engenharia de Computação

Segundo Semestre de 2023

Prova 2 – 8/11/2023

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. 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
```

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

```
public class Q1
{
    public static void main(String[] args)
    {
        int[] s = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}; // s[i] = i para todo i
        int[] t = StdIn.readAllInts(); // NUSP
        int M = s.length;
        int N = t.length;

        // opt[i][j] = length of LCS of s[i..M) and t[j..N)
        int[][] opt = new int[M + 1][N + 1];

        // compute length of LCS and all subproblems via dynamic programming
        for (int i = M - 1; i >= 0; i--)
            for (int j = N - 1; j >= 0; j--)
                if (s[i] == t[j])
                    opt[i][j] = opt[i + 1][j + 1] + 1;
                else
                    opt[i][j] = Math.max(opt[i + 1][j], opt[i][j + 1]);

        for (int i = 0; i < 10; i++) {
            for (int j = 0; j < N; j++)
                StdOut.print(opt[i][j] + " ");
            StdOut.println();
        }
        StdOut.println();

        int i = 0, j = 0;
        while(i < M && j < N) {
            if (s[i] == t[j]) {
                StdOut.print(s[i] + " ");
                i++;
                j++;
            }
            else if (opt[i + 1][j] >= opt[i][j + 1]) i++;
            else                                     j++;
        }
        StdOut.println();
    }
}
```

Rascunho

NUSP:								
0								
1								
2								
3								
4								
5								
6								
7								
8								
9								

Saída do programa

Q2 (3.0 pontos) Alice e Beto descobrem um tesouro com N itens valiosos. Cada item i ($0 \leq i < N$) tem um valor inteiro $val[i]$. Eles querem dividir o tesouro de forma que **cada um fique com exatamente o mesmo valor**. Eles querem saber quantas tais divisões justas/exatas existem, e escrevem o programa abaixo.

```
public class ExactSplitsPlain
{
    static int A, B, S;

    public static int exactSplits(int[] val, int[] a, int k) {
        if (k == a.length)
            if (A == B)
                return 1;
            else
                return 0;
        // início
        a[k] = 0;
        A += val[k];
        int x = exactSplits(val, a, k + 1);
        A -= val[k];
        a[k] = 1;
        B += val[k];
        int y = exactSplits(val, a, k + 1);
        B -= val[k];
        // fim
        return x + y;
    }

    public static int exactSplits(int[] val) {
        int[] a = new int[val.length];
        return exactSplits(val, a, 0);
    }

    public static void main(String[] args)
    {
        int[] val = StdIn.readAllInts();
        for (int i = 0; i < val.length; i++)
            S += val[i];
        StdOut.println("Valor a ser dividido: " + S);
        StdOut.println("Há " + exactSplits(val) + " divisões exatas");
    }
}
```

Alice e Beto percebem que é possível alterar o programa `ExactSplitsPlain.java` acima levemente para que ele fique mais rápido. O programa melhorado deles chama-se `ExactSplits.java`. Suponha que o programa `RandomValues.java` gere instâncias para os programas de Alice e Beto: suponha que, por exemplo, a execução

```
$ java-introcs RandomValues 5 1000 122
```

gere 5 inteiros aleatórios no intervalo $[1, 1000]$; o parâmetro 122 é usado como semente do gerador de números aleatórios em `RandomValues.java`. Alice e Beto ficam satisfeitos ao verem as seguintes execuções:

```
$ time java-introcs RandomValues 33 20 112023 | java-introcs ExactSplitsPlain
Valor a ser dividido: 384
Há 90123604 divisões exatas

real    0m34.359s
user    0m34.082s
sys     0m0.166s
$ time java-introcs RandomValues 33 20 112023 | java-introcs ExactSplits
Valor a ser dividido: 384
Há 90123604 divisões exatas

real    0m8.065s
user    0m8.024s
sys     0m0.087s
$
```

Nesta questão, você deve dizer como alterar o trecho entre "início" e "fim" em `ExactSplitsPlain.java` para obter um programa mais rápido como `ExactSplits.java`. Escreva abaixo seu trecho melhorado entre "início" e "fim" (o resto do programa deve ficar inalterado e não deve ser reproduzido aqui):

Q3 (4.0 pontos) Considere o seguinte programa incompleto:

```
public class CountInYoung
{
    // devolve o número de elementos estritamente
    // menores que x no Young tableau A
    public static int smaller(int[][] A, int x) {
        int N = A[0].length;   // número de colunas em A
        return smaller(A, x, 0, N - 1);
    }

    public static int smaller(int[][] A, int x, int i, int j) {
        ... // a completar
    }

    // devolve o número de elementos estritamente
    // maiores que x no Young tableau A
    public static int larger(int[][] A, int x) {
        int N = A[0].length;
        return larger(A, x, 0, N - 1);
    }

    public static int larger(int[][] A, int x, int i, int j) {
        ... // a completar
    }

    public static void main(String[] args)
    {
        int x = Integer.parseInt(args[0]);
        int M = Integer.parseInt(args[1]);
        int N = Integer.parseInt(args[2]);
        int[][] A = new int[M][N];

        for (int i = 0; i < M; i++)
            for (int j = 0; j < N; j++)
                A[i][j] = StdIn.readInt();

        // supomos que A é um Young tableau
        int s = smaller(A, x);
        int l = larger(A, x);
        int e = M * N - s - l;
        StdOut.println(s + " / " + e + " / " + l);
    }
}
```

O programa `CountInYoung.java` recebe na linha de comando um inteiro x e as dimensões M e N de uma matriz, que será dada na entrada padrão. Ademais, `CountInYoung.java` supõe que a matriz dada é um *Young tableau*: tanto suas linhas como suas colunas estão todas ordenadas de forma crescente (veja um exemplo abaixo).

Uma vez completo, `CountInYoung.java` pode ser usado da seguinte forma:

```
$ cat Young.txt
  5  23  54  67  89
  6  69  73  74  90
 10  71  84  84  94
 60  73  84  86  94
 91  91  92  93  94
$ java-introcs CountInYoung 73 5 5 < Young.txt
9 / 2 / 14
$
```

A saída `9 / 2 / 14` significa que o Young tableau 5×5 dado na entrada padrão contém 9 elementos menores que 73, dois de seus elementos são 73 e 14 elementos são maiores que 73.

Neste exercício, você deve completar o programa `CountInYoung.java`. Seu programa deve ser tal que, para produzir a saída, menos de $2(M + N)$ comparações envolvendo elementos do Young tableau de entrada A e x são feitas, onde M é o número de linhas em A e N é o número de colunas em A. Mais precisamente, `smaller()` deve fazer no total no máximo $M + N - 1$ tais comparações e `larger()` deve fazer no total no máximo $M + N - 1$ tais comparações. **Suas implementações de `smaller()` e `larger()` devem ser recursivas.**

Observação. O número de linhas em `A` é `A.length`.

(a) Implemente a função faltante `smaller()` em `CountInYoung.java` como descrito acima. Escreva a função completamente, incluindo sua assinatura (cabeçalho).

(b) Implemente a função faltante `larger()` em `CountInYoung.java` com descrito acima. Escreva a função completamente, incluindo sua assinatura (cabeçalho).

(c) Como explicitado acima, suas implementações de `smaller()` e `larger()` devem ser tais que elas fazem cada uma um total de no máximo $M + N - 1$ comparações entre elementos do Young tableau A e x. Argumente que este é de fato o caso para suas implementações. [**Sugestão.** Você pode achar conveniente considerar a quantidade $M + j - i$.]

(d) Suponha que $M = 1000$ e $N = 1000$. É claro que podemos fazer uma varredura da matriz A e comparar cada elemento de A com x, e contar quantos elementos de A são menores/iguais/maiores que x. Quantas comparações serão feitas nesse algoritmo elementar? Quantas comparações seu programa `CountInYoung.java` fará?

* * *

(Rascunho)