MAC0122 – Princípios de Desenvolvimento de Algoritmos

Engenharia de Computação

Segundo Semestre de 2023

Prova 1 – 13/9/2023

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 (3.0 pontos) Diga qual será a saída do programa abaixo quando executado com seu número USP como argumento de linha de comando.

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

```
public class Q1
{
    public static int lg(int N) {
        int t = 0;
        while (N >= 2) {
            N /= 2;
            t++;
        }
        return t;
    }

    public static long p(int e) {
        long r = 1;
        while (e > 0) {
            r *= 2;
            e--;
        }
        return r;
    }

    public static void main(String[] args)
    {
        int NUSP = Integer.parseInt(args[0]);

        int x = NUSP % 100000;
        int t = lg(x);
        StdOut.println(t);
        StdOut.println(p(t));
        StdOut.println(x);
        StdOut.println(p(t + 1));
        StdOut.println();

        x = NUSP / 100;
        t = lg(x);
        StdOut.println(t);
        StdOut.println(p(t));
        StdOut.println(x);
        StdOut.println(p(t + 1));
    }
}
```

Rascunho

Saída do programa

Q2 (3.0 pontos) Lembre que, para qualquer inteiro $k > 0$, podemos gerar um valor aleatório r uniformemente distribuído em $\{0, 1, \ldots, k-1\}$ fazendo

```
r = (int)(Math.random() * k);
```

Isto é, o valor de r após a execução da atribuição acima é i com probabilidade $1/k$ para todo i em $\{0, 1, \ldots, k-1\}$.

Suponha agora que $p_0, p_1, \ldots, p_{k-1}$ são reais não-negativos com $\sum_{0 \le i < k} p_i = 1$. Podemos então considerar o problema de gerar r com a distribuição definida por tais p_i. **Isto é, queremos gerar r de forma que a probabilidade de r ser i seja p_i para todo i em $\{0, 1, \ldots, k-1\}$.**

Escreva uma função com assinatura

```
public static int rand(double[] p)
```

que recebe no vetor p os valores $p_0, p_1, \ldots, p_{k-1}$ (isto é, p tem k elementos e $p[i] = p_i$ para todo i) e que devolve um valor r tal que $r = i$ com probabilidade p_i para todo i em $\{0, 1, \ldots, k-1\}$.

Sugestão. Você pode considerar e talvez usar a função auxiliar `psum()` abaixo.

```
public static double[] psum(double[] p) {
    double[] q = new double[p.length];
    double s = 0.0;
    for (int i = 0; i < p.length; i++) {
        s += p[i];
        q[i] = s;
    }
    return q;
}
```

Exemplo de uso. Sua função `rand()` poderia ser usada como na função `main()` abaixo.

```
public static void main(String[] args)
{
    int N = Integer.parseInt(args[0]);
    double[] p = StdIn.readAllDoubles();
    show(p);

    int[] f = new int[p.length];
    while (N > 0) {
        int r = rand(p);
        f[r]++;
        N--;
    }
    show(f);
}
```

A função `StdIn.readAllDoubles()` lê todos os `doubles` disponíveis na entrada padrão e monta um vetor com esses `doubles`. A função `show()` imprime as entradas do vetor fornecido.

Suponha que o arquivo `data.txt` contenha `0.5 0.3 0.2`. Seguem exemplos de execução de `main()` acima:

```
$ java-introcs Q2 100000 < data.txt
0.50 0.30 0.20
50013 30219 19768
$ java-introcs Q2 100000 < data.txt
0.50 0.30 0.20
50205 29978 19817
$ java-introcs Q2 100000 < data.txt
0.50 0.30 0.20
49734 30374 19892
$
```

Escreva aqui sua função `rand()` (escreva a função completa, isto é, não deixe de escrever sua assinatura (cabeçalho)). Seu algoritmo em `rand()` deve ser baseado no uso da função `Math.random()`. (Não use funções de `StdRandom.java`.)

Q3 (5.0 pontos) Considere a função `power()` abaixo.

```
public static long power(long 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 $b \geq 0$. Que valor é devolvido pela chama `power(a, b)`? (Ignore problemas de *overflow*.)

(d) Justifique sua resposta para o item (c) acima.

(e) Suponha que temos de pagar R$1 por cada multiplicação que é feita por `power()`. Quanto temos de pagar pela execução de `power(2, 5)`? Note que o número de `'.'` impressos por `power()` é quanto devemos pagar. Esboce como você chegou a sua resposta.

(f) Continuando com (e), quanto temos de pagar pela execução de `power(3, 32)`? Esboce como você chegou a sua resposta.

(g) Em MAC2166 você implementou uma função que calcula o mesmo valor que `power()` calcula. Em sua implementação de MAC2166, quantas multiplicações são feitas para se calcular o valor calculado por `power(3, 32)`? (O que se entende aqui por "implementação de MAC2166" é o algoritmo natural e mais simples para se calcular a função matemática calculada por `power()`.)

(h) Para um inteiro positivo b, seja $n(b)$ o número de bits necessários para se escrever b na base 2. Quanto temos de pagar para calcular `power(a, b)` no máximo, em termos de $n(b)$? Você consegue dizer o valor exato que devemos pagar em termos de $n(b)$ e algo mais? (Considere o número $u(b)$ de bits iguais a 1 na representação de b na base 2.) Justifique sua resposta.

* * *

(Rascunho)