# MAC0323 – Algoritmos e Estruturas de Dados II

Bacharelado em Ciência da Computação

Primeiro Semestre de 2024

Prova 2 – 25/6/2024

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.
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)** O programa abaixo computa o autômato do algoritmo KMP para uma dada entrada e imprime a "parte relevante" do autômato. Diga qual vai ser a saída deste programa quando executado com seu número USP e um certo parâmetro adicional $r$, onde $r = 3$. Por exemplo, se seu NUSP fosse 31415926, você teria de dizer qual é a saída da execução

```
$ java-algs4 KMPQ1 31415926 3
```

**Importante:** seu rascunho deve dar alguma indicação de como você chegou a sua resposta; por exemplo, desenhe o autômato.

**Importante:** seu NUSP não é 31415926.

```
import edu.princeton.cs.algs4.StdOut;

public class KMPQ1 {
    private final int R;       // the radix
    private final int M;       // length of pattern
    private int[][] dfa;       // the KMPQ1 automoton

    public KMPQ1(String pat) {
        this.R = 256;
        this.M = pat.length();

        dfa = new int[R][M];
        dfa[pat.charAt(0)][0] = 1;
        for (int X = 0, j = 1; j < M; j++) {
            for (int c = 0; c < R; c++)
                dfa[c][j] = dfa[c][X];     // Copy mismatch cases
            dfa[pat.charAt(j)][j] = j + 1; // Set match case
            X = dfa[pat.charAt(j)][X];     // Update restart state
        }
    }

    public int[][] dfa() {
        return dfa;
    }

    // s: string of digits
    // Returns a string with each digit of s reduced modulo r
    // Example: reduced("012345", 3) returns "012012"
    public static String reduce(String s, int r) {
        int M = s.length();
        String reduced = "";
        for (int i = 0; i < M; i++) {
            int d = s.charAt(i) - '0';
            reduced += (char)((d % r) + '0');
        }
        return reduced;
    }
```

```
    public static void main(String[] args) {
        String pat = args[0];
        int r = Integer.parseInt(args[1]);

        String reduced = reduce(pat, r);
        StdOut.println("Pattern: " + pat);
        StdOut.println("Reduced: " + reduced);

        KMPQ1 kmp = new KMPQ1(reduced);
        int[][] dfa = kmp.dfa();

        StdOut.println("DFA:");
        int M = reduced.length();
        StdOut.print("   ");
        for (int j = 0; j < M; j++)
            StdOut.printf(" %2c", pat.charAt(j));
        StdOut.print("\n   ");
        for (int j = 0; j < M; j++)
            StdOut.printf(" %2c", reduced.charAt(j));
        StdOut.println();

        for (int i = 0; i < r; i++) {
            StdOut.printf("%c: ", '0' + i);
            for (int j = 0; j < M; j++)
                StdOut.printf(" %2d", dfa['0' + i][j]);
            StdOut.println();
        }
    }
}
```

O começo da saída da execução de `java-algs4 KMPQ1 31415926 3` é dado abaixo:

```
$ java-algs4 KMPQ1 31415926 3
Pattern: 31415926
Reduced: 01112020
DFA:
     3  1  4  1  5  9  2  6
     0  1  1  1  2  0  2  0
0:   [...]
```

**Rascunho**

**Saída do programa**

**Q2 (3.0 pontos)** Considere o seguinte grafo dirigido com custos nos arcos $G$ com 11 vértices e 23 arcos:

```
0: 0->3  6.00  0->2  3.00  0->1  1.00
1: 1->4  1.00
2: 2->5  1.00
3: 3->9  1.00  3->8  1.00
4: 4->7  3.00  4->6  3.00  4->2  1.00
5: 5->9  3.00  5->6  1.00  5->3  1.00
6: 6->10  2.00  6->8  1.00  6->7  1.00
7: 7->10  3.00  7->5  1.00  7->3  1.00
8: 8->10  2.00  8->7  1.00  8->5  1.00
9: 9->10  1.00
10:
```

Assim, $G$ tem o arco $(0, 3)$ com custo 6, tem o arco $(0, 2)$ com custo 3, tem o arco $(0, 1)$ com custo 1, tem o arco $(1, 4)$ com custo 1, etc. Suponha que executamos o algoritmo de Dijkstra e descobrimos a distância distTo[$v$] do vértice 0 a todo vértice $v$ de $G$. Suponha que distTo[ ] é como a seguir:

```
distTo (from 0):
 0: 0.00
 1: 1.00
 2: 3.00
 3: 5.00
 4: 2.00
 5: 4.00
 6: 5.00
 7: 5.00
 8: 6.00
 9: 6.00
10: 7.00
```

Assim, descobrimos que, por exemplo, a distância de 0 a 10 em $G$ é 7. *Encontre todos os caminhos mais curtos de* 0 *a* 10 *em* $G$ (isto é, encontre todos os caminhos de 0 a 10 de custo 7).

**Sugestão:** (A) Seja $P$ um caminho mínimo de 0 a 10 e seja $e$ o último arco de $P$. Pense quais são as possibilidades para $e$. (B) Não desenhe o grafo $G$ todo. Ao pensar sobre (A), você perceberá que há um grafo menor $G'$ que é suficiente para você listar os caminhos mínimos pedidos ($G'$ tem 9 vértices e 11 arcos).

**Importante:** seu rascunho deve dar alguma indicação de como você chegou a sua resposta; por exemplo, diga a que conclusão você chegou ao pensar sobre (A) e desenhe $G'$, explicando como você descobriu $G'$.

Caminhos mínimos de 0 a 10:

Rascunho:

**Q3 (4.0 pontos)** Seja $D$ um grafo dirigido. Um vértice $v$ de $D$ é um *vértice do tipo A* se todo vértice de $D$ é acessível a partir de $v$ (existe um caminho dirigido de $v$ a $w$ para todo $w$ em $D$). Considere o seguinte programa (incompleto):

```
import edu.princeton.cs.algs4.StdOut;
import edu.princeton.cs.algs4.In;
import edu.princeton.cs.algs4.Digraph;
import edu.princeton.cs.algs4.BreadthFirstDirectedPaths;
[...]

public class Q3
{
    public static int typeAVertex(Digraph D) {

        [...]

    }

    public static void main(String[] args)
    {
        In in = new In(args[0]);
        Digraph D = new Digraph(in);
        int s = typeAVertex(D);
        if (s < 0)
            StdOut.println("No type A vertex");
        else {
            StdOut.println("Type A vertex: " + s);
            BreadthFirstDirectedPaths bfs = new BreadthFirstDirectedPaths(D, s);
            for (int v = 0; v < D.V(); v++) {
                StdOut.printf("%d to %d (%d):  ", s, v, bfs.distTo(v));
                for (int x : bfs.pathTo(v)) {
                    if (x == s) StdOut.print(x);
                    else        StdOut.print("->" + x);
                }
                StdOut.println();
            }
        }
    }
}
```

Preenchendo as partes faltantes adequadamente, podemos usar o programa acima para encontrar vértices do tipo A em digrafos. Se $D$ é o digrafo

```
4 vertices, 6 arcs
0: 1
1: 3 2
2: 0
3: 2 0
```

então, ao executarmos `Q3.java` com entrada $D$, podemos obter

```
Type A vertex: 0
0 to 0 (0):  0
0 to 1 (1):  0->1
0 to 2 (2):  0->1->2
0 to 3 (2):  0->1->3
```

Se $D'$ é o digrafo

```
4 vertices, 5 arcs
0: 3 2
1: 3 2
2:
3: 2
```

então, ao executarmos `Q3.java` com entrada $D'$, obtemos

```
No type A vertex
```

Nesta questão, você deve implementar a função faltante `typeAVertex()` (escreva a função completa, isto é, não omita sua assinatura). Sua implementação de `typeAVertex()` deve ter complexidade $O(V + E)$.

**Observação.** Há uma solução mais simples no caso em que o grafo de entrada $D$ é acíclico (isto é, quando $D$ é um DAG). Sua solução deve valer para grafos dirigidos arbitrários. Você pode optar por implementar uma solução que resolve o caso acíclico, valendo 2.0 pontos. Nesse caso, diga explicitamente que é essa sua opção: se você der uma solução para o caso geral que só funciona se $D$ é um DAG, sua solução será considerada incorreta e valerá pouco.

**Dica:** você pode achar conveniente lembrar as seguintes APIs:

```
public class DirectedDFS
------------------------------------------------------------------------
    DirectedDFS(Digraph G, int s)  ...  construtor / DFS a partir de s
    boolean marked(int v)          ...  v acessível a partir de s?
    int count()                    ...  número de vértices acessíveis

public class KosarajuSharirSCC
------------------------------------------------------------------------
    KosarajuSharirSCC(Digraph G)   ...  construtor / componentes fortes
    boolean stronglyConnected(int v, int w)
                                   ...  v e w mutuamente acessíveis?
    int count()                    ...  número de componentes fortes
    int id(int v)                  ...  identificador da componente
                                        forte de v (entre 0 e
                                        count() - 1)
```

A classe KosarajuSharirScc.java segue abaixo.

```
import edu.princeton.cs.algs4.Digraph;
import edu.princeton.cs.algs4.DepthFirstOrder;

public class KosarajuSharirSCC {
    private boolean[] marked;     // marked[v] = has vertex v been visited?
    private int[] id;             // id[v] = id of strong component containing v
    private int count;            // number of strongly-connected components

    public KosarajuSharirSCC(Digraph G) {
        DepthFirstOrder dfs = new DepthFirstOrder(G.reverse());
        marked = new boolean[G.V()];
        id = new int[G.V()];
        for (int v : dfs.reversePost()) {
            if (!marked[v]) {
                dfs(G, v);
                count++;
            }
        }
    }

    private void dfs(Digraph G, int v) {
        marked[v] = true;
        id[v] = count;
        for (int w : G.adj(v)) {
            if (!marked[w]) dfs(G, w);
        }
    }

    public int count() {
        return count;
    }

    public final boolean stronglyConnected(int v, int w) {
        return id[v] == id[w];
    }

    public int id(int v) {
        return id[v];
    }
}
```

(Rascunho)

(Rascunho)