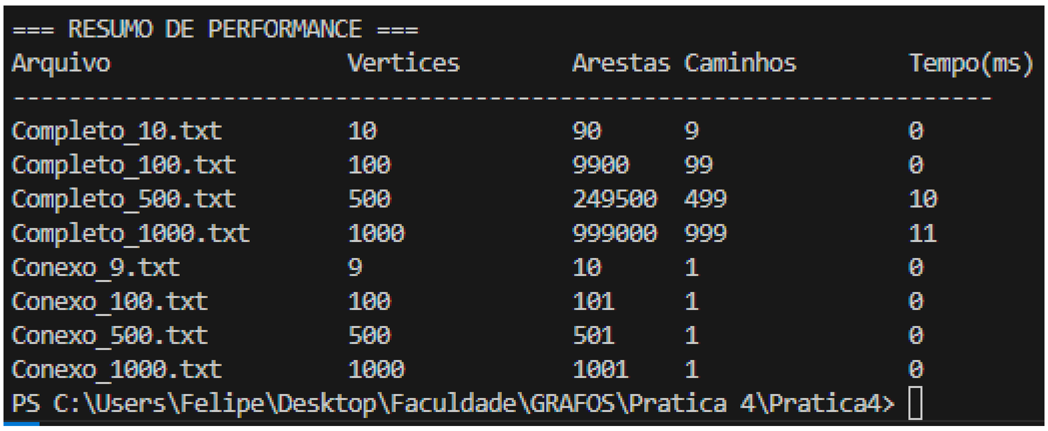
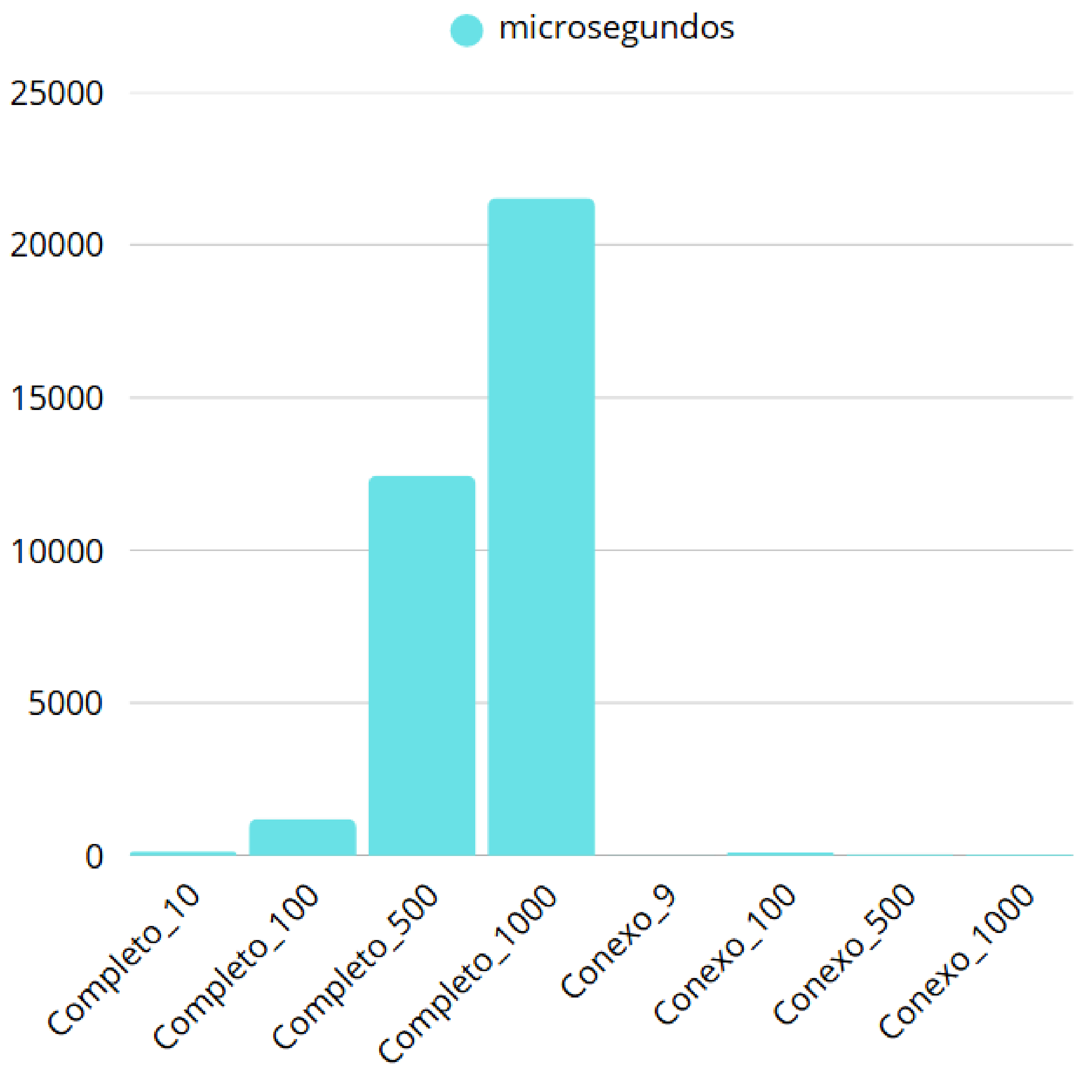
Implementação N.04 - Fluxo Máximo

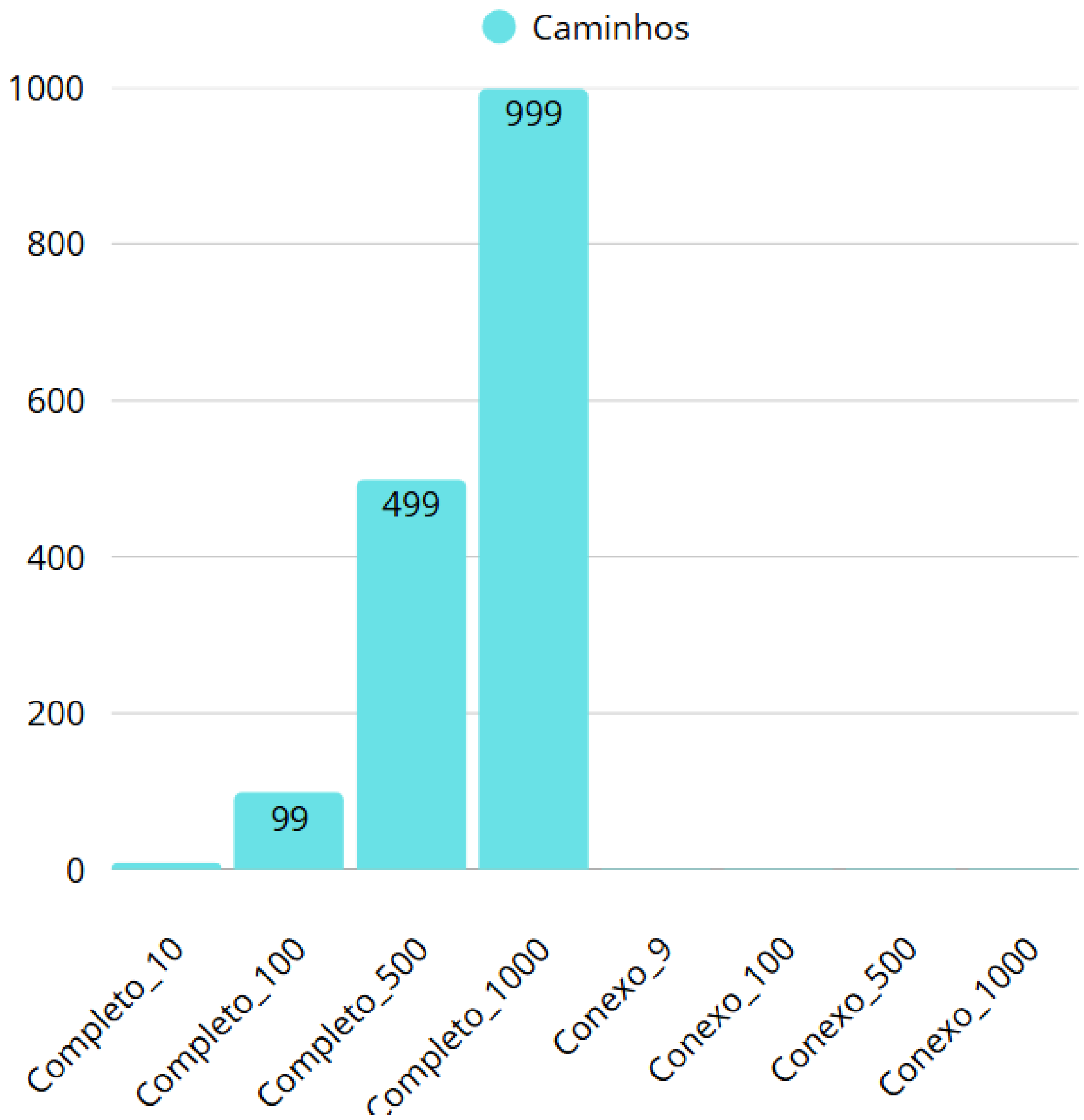


```
=== RESUMO DE PERFORMANCE ===
Arquivo                 Vertices        Arestas Caminhos        Tempo(ms)
----------------------------------------------------------------------
Completo_10.txt         10              90      9               0
Completo_100.txt        100             9900    99              0
Completo_500.txt        500             249500  499             10
Completo_1000.txt       1000            999000  999             11
Conexo_9.txt            9               10      1               0
Conexo_100.txt          100             101     1               0
Conexo_500.txt          500             501     1               0
Conexo_1000.txt         1000            1001    1               0
PS C:\Users\Felipe\Desktop\Faculdade\GRAFOS\Pratica 4\Pratica4>
```

Tempos de execução por tipo de grafo:



Número de caminhos encontrados:



Código:

```java
import java.io.*;
import java.util.*;

public class Main {
    private int n;
    private int[][] capacity;
    private List<List<Integer>> adj;
    private List<List<Integer>> paths;

    public Main(int vertices) {
        this.n = vertices;
        this.capacity = new int[n][n];
        this.adj = new ArrayList<>();
        this.paths = new ArrayList<>();

        for (int i = 0; i < n; i++) {
            adj.add(new ArrayList<>());
        }
    }

    public void addEdge(int u, int v, int cap) {
        if (capacity[u][v] == 0) {
            adj.get(u).add(v);
            adj.get(v).add(u);
        }
        capacity[u][v] += cap;
    }

    private boolean bfs(int source, int sink, int[] parent) {
        boolean[] visited = new boolean[n];
        Queue<Integer> queue = new LinkedList<>();
        queue.offer(source);
        visited[source] = true;
        parent[source] = -1;

        while (!queue.isEmpty()) {
            int u = queue.poll();

            for (int v : adj.get(u)) {
                if (!visited[v] && capacity[u][v] > 0) {
                    visited[v] = true;
                    parent[v] = u;
                    queue.offer(v);
                    if (v == sink) return true;
                }
            }
        }
        return false;
    }

    public int maxFlow(int source, int sink) {
        int[] parent = new int[n];
        int maxFlowValue = 0;
        paths.clear();

        while (bfs(source, sink, parent)) {
            int pathFlow = Integer.MAX_VALUE;

            // Encontrar capacidade mínima
            for (int v = sink; v != source; v = parent[v]) {
                int u = parent[v];
                pathFlow = Math.min(pathFlow, capacity[u][v]);
            }

            // Construir caminho atual
            List<Integer> currentPath = new ArrayList<>();
            for (int v = sink; v != source; v = parent[v]) {
                int u = parent[v];
                capacity[u][v] -= pathFlow;
                capacity[v][u] += pathFlow;
                currentPath.add(v);
            }
            currentPath.add(source);
            Collections.reverse(currentPath);
            paths.add(new ArrayList<>(currentPath));

            maxFlowValue += pathFlow;
        }

        return maxFlowValue;
    }

    public void printPaths(int source, int sink) {
        System.out.println("Caminhos disjuntos em arestas de " + source
+ " para " + sink + ":");
```

```
        System.out.println("Quantidade de caminhos: " + paths.size());

        for (int i = 0; i < paths.size(); i++) {
            System.out.print("Caminho " + (i + 1) + ": ");
            List<Integer> path = paths.get(i);
            for (int j = 0; j < path.size(); j++) {
                System.out.print(path.get(j));
                if (j < path.size() - 1) System.out.print(" -> ");
            }
            System.out.println();
        }
    }

    public void loadFromFile(String filename) {
        try (BufferedReader reader = new BufferedReader(new
FileReader(filename))) {
            String line = reader.readLine(); // primeira linha com
metadados

            while ((line = reader.readLine()) != null) {
                String[] parts = line.trim().split("\\s+");
                if (parts.length >= 3) {
                    int u = Integer.parseInt(parts[0]);
                    int v = Integer.parseInt(parts[1]);
                    addEdge(u, v, 1);
                }
            }
        } catch (IOException e) {
            System.err.println("Erro ao ler arquivo: " +
e.getMessage());
        }
    }

    public int[] getGraphInfo(String filename) {
        try (BufferedReader reader = new BufferedReader(new
FileReader(filename))) {
            String line = reader.readLine();
            String[] parts = line.trim().split("\\s+");
            int vertices = Integer.parseInt(parts[0]);
            int edges = Integer.parseInt(parts[1]);
            return new int[]{vertices, edges};
        } catch (IOException e) {
```

```java
            System.err.println("Erro ao ler arquivo: " +
e.getMessage());
            return new int[]{0, 0};
        }
    }

    public static void analyzeGraph(String filename) {
        System.out.println("\n=== ANÁLISE DO ARQUIVO: " + filename + "
===");

        Main tempGraph = new Main(10);
        int[] info = tempGraph.getGraphInfo(filename);

        if (info[0] > 0) {
            Main graph = new Main(info[0]);
            graph.loadFromFile(filename);

            long startTime = System.nanoTime();
            int maxPaths = graph.maxFlow(0, info[0] - 1);
            long endTime = System.nanoTime();

            long duration = (endTime - startTime) / 1000;

            graph.printPaths(0, info[0] - 1);
            System.out.println("Tempo de execução: " + duration + "
microsegundos");
            System.out.println("Vértices: " + info[0] + ", Arestas: " +
info[1]);
        } else {
            System.out.println("Erro ao processar arquivo: " +
filename);
        }
    }

    public static void runTests() {
        String[] files = {
            "Completo_10.txt",
            "Completo_100.txt",
            "Completo_500.txt",
            "Completo_1000.txt",
            "Conexo_9.txt",
            "Conexo_100.txt",
            "Conexo_500.txt",
            "Conexo_1000.txt"
        };

        System.out.println("\n" + "=".repeat(60));
        System.out.println("=== RESUMO DE PERFORMANCE ===");

System.out.println("Arquivo\t\t\tVertices\tArestas\tCaminhos\tTempo(ms)
");
        System.out.println("-".repeat(70));

        for (String filename : files) {
            Main tempGraph = new Main(10);
            int[] info = tempGraph.getGraphInfo(filename);

            if (info[0] > 0) {
                Main testGraph = new Main(info[0]);
                testGraph.loadFromFile(filename);

                long startTime = System.nanoTime();
                int paths = testGraph.maxFlow(0, info[0] - 1);
                long endTime = System.nanoTime();

                long duration = (endTime - startTime) / 1000000;

                System.out.printf("%-20s\t%d\t\t%d\t%d\t\t%d%n",
                                filename, info[0], info[1], paths,
duration);
            }
        }
    }

    public static void main(String[] args) {
        String[] files = {
            "Completo_10.txt",
            "Completo_100.txt",
            "Completo_500.txt",
            "Completo_1000.txt",
            "Conexo_9.txt",
            "Conexo_100.txt",
            "Conexo_500.txt",
            "Conexo_1000.txt"
        };
```

```java
        // Analisar cada grafo individualmente mostrando os caminhos
        for (String filename : files) {
            analyzeGraph(filename);
        }

        // Resumo de performance
        runTests();
    }
}
```