Departamento de Electrónica, Telecomunicações e Informática da Universidade de Aveiro

Algoritmos e Estruturas de Dados

2024/2025 — 1º Semestre

2nd Project — The GRAPH ADT

Deadline: January 3, at 18:00

The **GRAPH** abstract data type was presented during the lectures, and allows you to represent and operate on **graphs and directed graphs**, with or without weights associated with their edges. The data structure consists of a **list of vertices** and, for each vertex, its **list of adjacencies**. These lists are defined using the generic data type **SORTED-LIST**.

The **GRAPH** abstract data type only provides the **basic operations** on graphs. Other **algorithms** are implemented in **stand-alone modules**.

OBJECTIVES

This project will allow the development of **algorithms** on **graphs without weights associated with their edges**, and the **analysis of the computational efficiency** of some of the developed strategies.

It is required to:

1. **GRAPH ADT:** develop the function which, given a directed graph, creates the corresponding **transposed directed graph**.

2. **BELLMAN-FORD Module:** develop the function which, given a graph, with no weights associated with its edges, and an initial vertex, constructs the **the shortest-paths tree** between that initial vertex and each of the other reachable vertices, using the **Bellman–Ford algorithm**. Consult, for example, Wikipedia.

3. **TRANSITIVE-CLOSURE Module:** develop the function which allows, given a directed graph, without weights associated with its edges, to create the directed graph that is its **transitive closure**. This graph has the same vertices as the original graph, and there is an oriented edge between vertices u and v if, in the original graph, v is reachable from u, i.e., there is a directed path between those vertices. The **reachable vertices** must be determined using the **BELLMAN-FORD module**.

4. **ALL-PAIRS-SHORTEST-DISTANCES Module:** develop the functionalities which allow, given a directed graph, without weights associated with its edges, to compute the **distance matrix** that, for each **pair of vertices**, contains the distance associated with the corresponding **shortest path**, if any. The shortest paths must be determined using the **BELLMAN-FORD module**.

5. **Extra module – Additional grading**

   ECCENTRICITY-MEASURES Module: develop the functionalities which allow, given a directed graph, without weights associated with its edges, to compute graph and vertex features, which are based on the distance values associated with shortest paths. The following must be computed: 1) **the eccentricity of each vertex** (i.e., the greatest distance between that vertex and each of the other vertices of the graph), 2) **the graph radius** (i.e., the smallest eccentricity value of all its vertices), 3) **the graph diameter** (i.e., the largest eccentricity value of all its vertices), and 4) **the set of central vertices** (i.e., the set of vertices whose eccentricity value is equal to the radius of the graph). The distances between pairs of vertices must be determined using the **ALL-PAIRS-SHORTEST-DISTANCES module**.

- **Complexity Analysis:** Characterize the algorithmic complexity of the solutions implemented for 1) the **Bellman-Ford algorithm**, and for 2) the **algorithm that constructs the transitive closure** of a graph.

BASIC TASKS

- **Complete the development of the required functions in the various `.c` files.**
- **The corresponding `.h` files must not be changed.**
- Ensure that those functions perform correctly for the (very simple) directed graphs of the sample files: **`Test*.c`**
- Test the algorithms with more complex oriented graphs.

COMPLEXITY ANALYSIS – FOR GRADES HIGHER THAN 16 POINTS

- Write a **short report** (max. 6 pages), with the characterization of the **computational complexity** of the strategies implemented for 1) the **Bellman-Ford algorithm**, and for 2) the **algorithm constructing the transitive closure** of a graph.

1. The report should include a brief description of the **metric(s) used** to measure computational complexity and **tables (graphs) with the results of the tests** performed.

1. **The delivery of a report does not in itself mean that the project grade will be higher than 16 points.**

Attention – Code Development

- The vertices of a graph are sequentially numbered: 0, 1, 2, ...
- You must respect the function prototypes defined in the various header files.
- You can create **auxiliary functions (static)** whenever you find it useful.

- The **code** developed should be **clear** and **commented** on appropriately: the identifiers chosen for the variables and the structure of the code, as well as any comments, should be sufficient to understand it.

- At the end of the project, you have to deliver a **ZIP file** with the developed code and a **PDF file** with the complexity analysis report.

- No report on code development is required.

Grading Criteria

- **Development and testing of the requested functions (16 points)**
  - Code quality (efficiency, readability, clarity, robustness)
  - Code testing and memory leak checking

- **Report (4 points)**
  - Overview/Presentation/Conclusion
  - Complexity of the two algorithms analyzed
    - Experimental data

Note the following:

- The project is to be carried out in groups of 2 students, keeping the groups from the previous project, whenever possible.

- The project delivery (code + optional report) will be done through the eLearning platform.