Practical Work #1

Graph Algorithms

Pricop Laurențiu, 916/1
Year 1 2021/2022

Specification

The interface `IDirectedGraph<TVertex, TCost>` defines the required operations for a directed graph.

`TVertex` and `TCost` are type parameters that represent a vertex, respectively a cost. When using the class, they can take any value. In my practical examples, they are both assigned to `int`.

The `IDirectedGraph<TVertex, TCost>` interface defines the following operations:

- `int VertexCount { get; }`
    - Returns the number of vertices in the graph.
- `IEnumerable<TVertex> Vertices { get; }`
    - Returns an `IEnumerable` of all vertices in the graph.
- `bool IsVertex(TVertex v)`
    - Returns `true` if `v` is a valid vertex.
- `bool IsEdge(TVertex start, TVertex end)`
    - Returns `true` if:
        - `start` and `end` are valid vertices, and
        - (`start`, `end`) is an existing edge.
- `int InDegree(TVertex v)`
    - **Precondition:** `v` is a valid vertex.
    - Returns the in degree of vertex `v`.
- `int OutDegree(TVertex v)`
    - **Precondition:** `v` is a valid vertex.
    - Returns the out degree of vertex `v`.
- `IEnumerable<TVertex> InboundVerticesOf(TVertex v)`
    - **Precondition:** `v` is a valid vertex.
    - Returns an `IEnumerable` of all vertices that are on inbound edges to `v`.
- `IEnumerable<TVertex> OutboundVerticesOf(TVertex v)`
    - **Precondition:** `v` is a valid vertex.

  - Returns an `IEnumerable` of all vertices that are on outbound edges from `v`.

- `TCost GetCostFor(TVertex s, TVertex e)`
  - **Precondition**: (`s`, `e`) is a valid edge.
  - Returns the cost for the (`s`, `e`) edge.

- `void SetCostFor(TVertex v, TVertex e, TCost c)`
  - **Precondition:** (`s`, `e`) is a valid edge.
  - Sets the cost of the (`s`, `e`) edge to `c`.

- `void AddVertex(TVertex v)`
  - **Precondition:** `v` does not exist as a vertex already.
  - Adds `v` as a new vertex in the graph.

- `void RemoveVertex(TVertex v)`
  - **Precondition:** `v` is a valid vertex.
  - Removes vertex `v` from the graph, along with all associated edges.

- `void AddEdge(TVertex v1, TVertex v2, TCost cost)`
  - Adds (`v1`, `v2`) as a new edge in the graph with cost `cost`.
  - If any of the vertices do not exist, they will be added.

- `void RemoveEdge(TVertex v1, TVertex v2)`
  - **Precondition:** (`v1`, `v2`) is a valid edge.
  - Removes edge (`v1`, `v2`) from the graph.

- `IDirectedGraph<TVertex, TCost> Copy()`
  - Returns a copy of the graph.

Utility methods

Static class `GraphUtils` defines the following utility methods:

- `void ToFile(string filename, IDirectedGraph<int, int> graph)`
  - Writes the `graph` to the file at `filename`.

- `IDirectedGraph<int, int> FromFile(string filename)`
  - **Precondition:** `filename` exists as a file.
  - Reads the contents of the file and creates a graph based on it.

- `IDirectedGraph<int, int> NewRandom(int vertices, int edges)`
  - **Precondition:** `vertices`^2 is greater than or equal to `edges`.
  - Creates a graph with `vertices` vertices and `edges` edges, randomly.
  - Failure to meet above precondition results in a `GraphException`.

Implementation

The class `DirectedGraph<TVertex, TCost>` implements the interface defined above.

All preconditions from the interface are explicity checked for, and a `GraphException` with an appropriate message is thrown when a precondition is not met.

Additionally, the class defines two constructors:

- `DirectedGraph()`
    - Creates a graph with no vertices and edges.
- `DirectedGraph(List<TVertex> vertices, List<(TVertex, TVertex, TCost)> edges)`
    - Creates a graph that has the elements from `vertices` as vertices, and the edges defined in `edges` as tuples of (from, to, cost).