**class directed graph :**

* **constructor / copy constructor / destructor**
* **3 dictionaries:**
  + **din : stores every vertex as a unique key and all its inbound neighbours in a list assigned to the corresponding key**
  + **dout : stores every vertex as a unique key and all its outbound neighbours in a list associated to the corresponding key**
  + **dcost : stores a pair of 2 vertices (source vertex, target vertex) – the edge as a unique key and associated to it is the cost**
* **all the specific methods required for a directed graph**

**class ui :**

* **constructor / copy constructor / destructor**
* **2 directed graphs:**
  + **main\_graph : used in almost all operations**
  + **copy\_graph : used when a copy is desired or when a random graph is generated**
* **reading from a file (in 2 different ways)**
* **saving the information of a graph to a file**
* **generating a random graph**
* **handle edge string**
* **menu :**

1. **Read from a file :**

* **a filename is provided by the user**
* **has 2 reading formats (with the given number of vertices and edges / without)**
* **if neither one of these is provided, a message is printed otherwise the reading is performed and the information is stored in the main graph**
* **complexity : Theta(nr of vertices + nr of edges)**

1. **Save the graph to a file :**

* **a filename is provided by the user**

**class graph :**

* **constructor / copy constructor / destructor**
* **3 dictionaries:**
  + **din : stores every vertex as a unique key and all its inbound neighbours in a list assigned to the corresponding key**
  + **dout : stores every vertex as a unique key and all its outbound neighbours in a list assigned to the corresponding key**
  + **dcost : stores a pair of 2 vertices (source vertex, target vertex) – the edge as a unique key and assign to it is the cost**
* **all the specific methods required for a directed graph**

**class ui :**

* **constructor / copy constructor / destructor**
* **reading from a file (in 2 different ways)**
* **saving the information of a graph to a file**
* **generating a random graph**
* **menu :**

1. **Read from a file :**

* **a filename is provided by the user**
* **has 2 reading formats (with the given number of vertices and edges / without)**
* **if neither one of these is provided, a message is printed otherwise the reading is performed and the information is stored in the main graph**
* **complexity : Theta(nr of vertices + nr of edges)**

1. **Saving the graph to a file:**

* **a filename is provided by the user**
* **the information stored in the graph is outputted in the provided textfile**
* **complexity : Best Case – Theta(nr of vertices) / Worst Case - Theta(nr of vertices \* nr of edges) / Total complexity – O(nr of vertices \* nr of edges**

CAMPEAN CATALIN ALEXANDRU 912/1

ASSIFNMENT 1 DOCUMENATTION C++

* **the formation stored in the graph is outputted in the provided**

**textfile**

* **complexity : Best Case – Theta(nr of vertices) / Worst Case - Theta(nr of vertices + nr of edges) / Total complexity – O(nr of vertices \* nr of edges**

1. **Get number of vertices :**

* **returns the number of vertices of the graph**
* **complexity : Theta(1)**

1. **Get number of edges :**

* **returns the number of edges of the graph**
* **complexity : Theta(1)**

1. **Get all vertices :**

* **returns a list containing all the vertices of the graph**
* **complexity : Theta(nr of vertices)**

1. **Get all edges :**

* **returns a list containing all the edges of the graph**
* **complexity : Theta(nr of edges)**

1. **Get in / out degree of a vertex :**

* **the desired vertex is provided by the user**
* **precondition : checks if the vertex exists, raises an exception if not**
* **returns the in and out degree of the desired vertex**
* **complexity : Theta(1)**

1. **Get inbound neighbours of a vertex :**

* **the desired vertex is provided by the user**
* **precondition : checks if the vertex exists, raises an exception if not**
* **returns the number of vertices of the graph**
* **complexity : Theta(1)**

1. **Get outbound neighbours of a vertex:**

* **the desired vertex is provided by the user**
* **precondition : checks if the vertex exists, raises an exception if not**

ASSIFNMENT 1 DOCUMENATTION C++

* **returns the number of vertices of the graph**
* **complexity : Theta(1)**

1. **Get cost :**

* **the desired edge is provided by the user**
* **precondition : checks if the edge exists, raises an exception if not**
* **returns the cost associated to the desired edge**
* **complexity : Theta(1)**

1. **Find if vertex exists :**

* **the desired vertex is provided by the user**
* **returns true if the vertex exists or false otherwise**
* **complexity : Theta(1)**

1. **Find if vertex exists :**

* **the desired vertex is provided by the user**
* **returns true if the vertex exists or false otherwise**
* **complexity : Theta(1)**

1. **Add vertex :**

* **the desired vertex is provided by the user**
* **precondition : checks if the vertex already exists in the graph, raises an exception if it does**
* **returns true if the vertex was added or false otherwise**
* **complexity : Theta(1)**

1. **Delete vertex :**

* **the desired vertex is provided by the user**
* **precondition : checks if the vertex exists in the graph, raises an exception if not**
* **returns true if the vertex was deleted or false otherwise**
* **complexity : Theta(nr of inbound neighbours + nr of outbound neighbours)**

1. **Add edge :**

* **the desired edge along with the cost is provided by the user**
* **precondition : checks if the edge already exists in the graph, raises an exception if it does**

ASSIFNMENT 1 DOCUMENATTION C++

ASSIFNMENT 1 DOCUMENATTION C++

* **returns true if the vertex was added or false otherwise**
* **complexity : Theta(1)**

1. **Delete edge :**

* **the desired edge is provided by the user**
* **precondition : checks if the edge exists in the graph, raises an exception if not**
* **returns true if the edge was deleted or false otherwise**
* **complexity : Theta(1)**

1. **Update edge information :**

* **the desired edge and the new info is provided by the user**
* **precondition : checks if the edge exists, raises an exception if not**
* **returns the old and the new information associated to the desired edge**
* **complexity : Theta(1)**

1. **Print graph :**

* **Has 2 options (printing the main graph or the copy graph)**
* **complexity : Best Case – Theta(nr of vertices) / Worst Case – Theta(nr of vertices + nr of edges) / Total Complexity – O(nr of vertices + nr of edges)**

1. **Copy graph :**

* **copies the main graph into the copy graph**
* **complexity : Theta(1)**

1. **Create random graph :**

* **a filename is provided by the user (for the output)**
* **the nr of vertices and the number of edges are provided by the user**
* **precondition : checks if the number of edges is less than or equal to the number of vertices squared raises an exception if not**
* **the copy is stored in the output file as well as in the copy graph**
* **complexity : Theta(nr of vertices + nr of edges)**